BIALKE – BERENDSEN –
GLISZCZYNSKI

Alles über den MZ 700

Das große Buch zum Sharp MZ 700 Personalcomputer

Ein BBG - Software - Buch

BIALKE – BERENDSEN – GLISZCZYNSKI

Software

Alles über den

MZ 700



BBG Software
Schimmelmanstr. 90
2070 Ahrensburg

2. Auflage

Vorwort

Dieses Buch ist als ein erweitertes Programmierhandbuch für den SHARP MZ-700 Computer gedacht.Dieses Buch kann jedoch auch von MZ-80K und MZ-80A Besitzern benutzt werden,da diese Computer weitgehenst kompatibel zum MZ-700 sind.Es können allerdings nicht alle Informationen dieses Buches auf den MZ-80 übertragen werden.

Des öfteren wird in diesem Buch auf das SHARP-Bedienungshandbuch verwiesen. Befehle und Kapitel,die in diesem Bedienungshandbuch ausreichend beschrieben sind,werden hier nicht mehr behandelt.

Sämtliche Kapitel in diesem Buch sind selbstständig,d.h. Sie sind nicht an die vorgegebene Reihenfolge gebunden.

Wenn Sie an einem kostenlosen Soft- und Hardwareprospekt unserer Firma interessiert sein sollten,senden Sie bitte die vorgefertigte Anfragekarte am Ende des Buches an uns.

Wir bedanken uns hiermit auch für die Unterstützung der Firma SHARP ELECTRONIC (EUROPE) GmbH.
Vielen Dank auch an die Firma Jölo für drucktechnische Hinweise.

BBG Software :

Axel Bialke

Stephan Berendsen

Andreas v. [illegible]

Inhaltsverzeichnis

MZ-700 BASIC INTERPRETER 1Z-013B V.1.0A

Dieser Abschnitt des Buches beschäftigt sich mit dem Basic Interpreter des MZ-700.BASIC bedeutet 'beginners all purpose symbolic instruction code'.
Es ist eine höhere Programmiersprache und wird in den Computer mittels des Cassettenrekorders eingeladen.
Der gesamte Ladevorgang des etwa 28000 Byte langen Basics dauert etwa 3 1/2 Minuten.
Das Basic wird ab Adresse 1200H bis 7DA0H während des Ladevorganges abgelegt.Dann startet sich das Basic und transferiert sich in den Bereich von 0000H bis 6B9AH.Von 0000H bis 1000H liegt der Basicmonitor im nachfolgenden auch RAM Monitor genannt.Der alte ROM Monitor wird weggeschaltet.
Auf das Basic selbst soll hier nicht weiter eingegangen werden,denn dieses ist,unserer Meinung nach,im Handbuch von Sharp sehr gut erklärt.
Vielmehr soll hier in diesem Abschnitt die interne Arbeitsweise dargestellt werden und dem Anwender einige Tricks und Manipulationen vermittelt werden.
einige Befehle wurden im Handbuch von Sharp überhaupt nicht erwähnt.
Deshalb soll dieses hier kurz geschehen.

TRON Dieser Befehl bewirkt,das auf dem Bildschirm die Zeilennummer ausgegeben wird,die der Interpreter gerade abarbeitet.
Dieses kann vor allen bei der Fehlersuche und Programmerstellung manchmal recht nützlich sein.

TROFF Mit diesem Befehl wird der oben angeführte TRON Befehl wieder abgeschaltet.

POKE Dieser Befehl wird im Handbuch zwar erwähnt,aber er ist auch mit mehreren Parametern möglich z.B. POKE53248,65,66,67
Es bewirkt das gleiche wie: POKE53248,65:POKE53249,66:POKE53250,67

JOY Dieser Befehl kontrolliert den Joystick,der an den vorgesehenen Eingängen angeschloßen werden kann.
Handhabung des Befehls liegt dem Joystick bei.

CLS Dieser Befehl löscht den ganzen Bildschirm und bewegt den Cursor in die obere linke Ecke.Wie die Taste 'CLR'.

HEX$ Dieser Befehl rechnet eine Dezimale Zahl in eine Hexadezimale um.
Z.B. PRINTHEX$(255) ergibt FFH.

1.1 Organisation einer Basiczeile im Speicher

Eine Basiczeile wird im Speicher nicht so abgelegt,wie sie auf dem Bildschirm erscheint,sondern wird in Token zerlegt.Jedem Basicbefehl wird ein Hexcode zugeordnet,der ein oder zwei Byte umfaßt.
Dieses hat den Vorteil,daß jeder Befehl nur maximal 2 Byte benötigt (die meisten brauchen nur einen).
Das Basicprogramm beginnt immer ab Adresse 6BCFH.
Ein Basiczeile ist wie folgt strukturiert:

1) Die ersten beiden Byte bilden die Anzahl der Zeichen in der Basiczeile. Eine Basiczeile kann maximal 255 Byte lang sein.Dieses ergibt sich aus der Bedingung,daß das zweite Byte (High Byte) immer 0 sein muß.

2) Das dritte und das vierte Byte bilden die Zeilennummer der Basiczeile.

3) Die folgenden Bytes bis zum End Byte sind Token,ASCII Kode,Variable oder Strings

4) Das End Byte ist 0.Dieses zeigt das Ende der Basiczeile an.

5) Wenn die Zeilennummer und der Zeichenzähler 0 sind,dann ist das Basicprogramm zu Ende.

1.1.1 Beispiel eines Programms in Basic und die Ablage im Speicher.

Bildschirm:

```
10 REM BBG-DEMO
20 FORA=1TO255
25 PRINTCHR$(A);:NEXT
30 PRINT"Info Basic"
40 GOTO10
60 END
```

Speicher:

Adresse	Inhalt	Bedeutung
6BCF	0F	Anzahl der Bytes in der Zeile (low Byte)
6CD0	00	Anzahl der Bytes in der Zeile (high Byte)
6CD1	0A	Zeilennummer (low Byte) = 10
6CD2	00	Zeilennummer (high Byte)
6CD3	97	REM
6CD4	20	'SPACE'
6CD5	42	B
6CD6	42	B
6CD7	47	G
6CD8	2D	-
6CD9	44	D
6CDA	45	E
6CDB	4D	M
6CDC	4F	O
6CDD	00	Ende der Basiczeile
6CDE	15	Anzahl der Bytes in der Zeile (low Byte)
6CDF	00	Anzahl der Bytes in der Zeile (high Byte)
6CE0	14	Zeilennummer (low Byte) = 20
6CE1	00	Zeilennummer (high Byte)
6CE2	8D	FOR
6CE3	41	A
6CE4	F4	=
6CE5	15	Konstantenbuffer
6CE6	81	Adresse 1581H
6CE7	00	
6CE8	00	
6CE9	00	
6CEA	00	
6CEB	E0	TO
6CEC	15	Konstantenbuffer
6CED	88	Adresse 1588H
6CEE	7F	
6CEF	00	
6CF0	00	
6CF1	00	
6CF2	00	Ende der Basiczeile

6CF3	0E	Anzahl der Bytes in der Zeile (low Byte)
6CF4	00	Anzahl der Bytes in der Zeile (high Byte)
6CF5	19	Zeilennummer (low Byte) = 25
6CF6	00	Zeilennummer (high Byte)
6CF7	8F	PRINT
6CF8	FF	1. Token CHR$
6CF9	A0	2. Token CHR$
6CFA	28	(
6CFB	41	A
6CFC	29	)
6CFD	3B	;
6CFE	3A	:
6CFF	8E	NEXT
6D00	00	Ende der Basiczeile
6D01	12	Anzahl der Bytes in der Zeile (low Byte)
6D02	00	Anzahl der Bytes in der Zeile (high Byte)
6D03	1E	Zeilennummer (low Byte) = 30
6D04	00	Zeilennummer (high Byte)
6D05	8F	PRINT
6D06	22	"
6D07	49	I
6D08	B0	n
6D09	AA	f
6D0A	B7	o
6D0B	20	'SPACE'
6D0C	42	B
6D0D	A1	a
6D0E	A4	s
6D0F	A6	i
6D10	9F	c
6D11	22	"
6D12	00	Ende der Basiczeile
6D13	09	Anzahl der Bytes in der Zeile (low Byte)
6D14	00	Anzahl der Bytes in der Zeile (high Byte)
6D15	28	Zeilennummer (low Byte) = 40
6D16	00	Zeilennummer (high Byte)
6D17	80	GOTO
6D18	0C	

6D19	CF	1. Byte der absoluten Adresse	6BCFH = Zeile 10
6D1A	6B	2. Byte der absoluten Adresse	6BCFH = Zeile 10
6D1B	00	Ende der Basiczeile	
6D1C	06	Anzahl der Bytes in der Zeile (low Byte)	
6D1D	00	Anzahl der Bytes in der Zeile (high Byte)	
6D1E	3C	Zeilennummer (low Byte) = 60	
6D1F	00	Zeilennummer (high Byte)	
6D20	98	END	
6D21	00	Ende der Basiczeile	
6D22	00	ENDE Zeilenzähler = 0	
6D23	00	ENDE Zeilennummer = 0	

1.2 Basicbefehle,Adresse und Token

Hier wird noch einmal eine Aufstellung von allen Basicbefehlen,deren Token und den dazugehörigen Einsprungadressen der Routinen gegeben.
Dieses könnte vor allem den Anwendern helfen,die einzelne Basic-Routinen ansehen oder verändern wollen.
Einige der Basicbefehle sind in der Tokenliste zwar enthalten,führen aber nur auf 'SYNTAX ERROR'.
Diese Kommandos sind hauptsächlich für die Diskette und wurden aus Platzgründen nicht implementiert.
Die Befehle sind nach den Token sortiert.
Alle Adressenangaben in der Tabelle sind Hexadezimal zu verstehen.
Die Math.-Routine ist eine Routine,die Arithmetische Ausdrücke in Konstante umrechnet.Diese Routine liegt ab 528B.

Befehl	Token	Routineinsprung	Bemerkung
GOTO	80	3807	
GOSUB	81	36CD	
RUN	83	1C6E	
RETURN	84	3694	
RESTORE	85	25A8	
RESUME	86	36FB	
LIST	87	4102	
DELETE	89	3456	
RENUM	8A	3471	
AUTO	8B	21D5	
FOR	8D	1CBF	
NEXT	8E	1D6F	
PRINT	8F	1E6A	
INPUT	91	22CA	
IF	93	383F	
DATA	94	3323	wie REM
READ	95	25E3	
DIM	96	5AEA	
REM	97	3323	wie DATA
END	98	21B7	
STOP	99	2071	
CONT	9A	209F	
CLS	9B	38CE	

ON	9D	3792	
LET	9E	1959	
NEW	9F	2248	
POKE	A0	33D7	
OFF	A1	20FE	Error
MODE	A2	4DD0	
SKIP	A3	4E2C	
PLOT	A4	3A31	
LINE	A5	4E8C	
RLINE	A6	4EFE	
MOVE	A7	4F05	
RMOVE	A8	4F14	
TRON	A9	227E	
TROFF	AA	2282	
INP	AB	3350	
GET	AD	3389	
PCOLOR	AE	4F1B	
PHOME	AF	4F50	
HSET	B0	4F64	
GPRINT	B1	4F76	
KEY	B2	4321	
AXIS	B3	4FFC	
LOAD	B4	41D1	
SAVE	B5	42A4	
MERGE	B6	41AB	
CONSOLE	B8	39CB	
OUT	BA	3339	
CIRCLE	BB	5050	
TEST	BC	5227	
PAGE	BD	523B	
ERASE	C0	20FE	Error
ERROR	C1	2105	
USR	C3	3305	
BYE	C4	13C2	
DEF	C7	251B	
WOPEN	CE	46A3	
CLOSE	CF	4545	
ROPEN	D0	46DF	
KILL	D9	20FE	Error
TO	E0	----	FOR-Routine
STEP	E1	----	FOR-Routine

THEN	E2	----	IF-Routine
USING	E3	----	PRINT-Routine
TAB	E6	----	PRINT-Routine
SPC	E7	----	PRINT-Routine
OR	EB	20FE	Error
AND	EC	20FE	Error
><	EE	----	Math.-Routine
<>	EF	----	Math.-Routine
=<	F0	----	Math.-Routine
<=	F1	----	Math.-Routine
=>	F2	----	Math.-Rcutine
>=	F3	----	Math.-Routine
=	F4	----	Math.-Routine
>	F5	----	Math.-Routine
<	F6	----	Math.-Routine
+	F7	----	Math.-Routine
-	F8	----	Math.-Routine
/	FB	----	Math.-Routine
*	FC	----	Math.-Routine
↑	FD	----	Math.-Routine
SET	FE 81	3902	
RESET	FE 82	3935	
COLOR	FE 83	4473	
MUSIC	FE A2	443C	
TEMPO	FE A3	4463	
CURSOR	FE A4	336C	
VERIFY	FE A5	42D1	
CLR	FE A6	224E	
LIMIT	FE A7	340B	
BOOT	FE AE	3A6A	Sprung nach 0000
INT	FF 80	6277	
ABS	FF 81	6272	
SIN	FF 82	63C6	
COS	FF 83	63B0	
TAN	FF 84	64BD	
LN	FF 85	6736	
EXP	FF 86	6615	
SQR	FF 87	62D0	
RND	FF 88	65D0	
PEEK	FF 89	65B5	
ATN	FF 8A	62EE	

SGN	FF 8B	657A	
LOG	FF 8C	672A	
PAI	FF 8E	65A2	
RAD	FF 8F	659D	
EOF	FF 95	20FE	Error
JOY	FF 9E	39A3	
STR$	FF A1	5677	
HEX$	FF A2	55EC	
ASC	FF AB	56A0	
LEN	FF AC	56AC	
VAL	FF AD	56B4	
ERN	FF B3	5562	
ERL	FF B4	556A	
SIZE	FF B5	5548	
LEFT$	FF BA	56C7	
RIGHT$	FF BB	56E4	
MID$	FF BC	5702	
TI$	FF C4	573F	
FN	FF C7	5CF7	

1.3 Manipulationen im Basic Monitor mit PEEK & POKE

Dieser Abschnitt beschäftigt sich mit dem Basic Monitor,welcher im Speicher von 0000H bis 0FFFH liegt.Es werden spezielle Adressen angegeben, deren Veränderung zur Erzeugung von speziellen Effekten dienen können. Diese Speicherzellen können entweder mit POKE oder USR verändert werden. Die Speicherzellen werden jeweils Dezimal und Hexadezimal angegeben.

Hex	Dezimal	Bedeutung
0015	21	Anzeige eines String auf dem Bildschirm. z.B. USR(21,B$) ; B$ wird ab Cursorposition ausgegeben.
0030	48	Spielen eines Musikstring. z.B. USR(48,M$) ; Spielt M$.
003E	62	USR(62) erzeugt einen kurzen Ton.
0044	68	USR(68) schaltet den Tongenerator an. Tonfrequenz über 2717 und 2718 steuerbar.
0047	71	USR(71) schaltet den Tongenerator wieder ab.
004D	77	PLOTTER an/aus. Wenn der Inhalt 1 ist,dann erfolgen alle Ausgaben auf den Drucker.Bei 0 auf den Bildschirm.
004E	78	CONSOLE an/aus. Wenn der Inhalt 1 ist,dann wurde bereits Parameter bei CONSOLE eingegeben.Bei 0 nicht.
0054	84	Diese Adresse beinhaltet die momentane Cursorspalte (0-39) POKE84,20 bewegt den Cursor an die 20 Spalte. Wie TAB(20).
0055	85	Hier wird die aktuelle Cursorposition ausgelesen. (0-24) POKE85,7 z.B. bewegt den Cursor in die 7 Zeile gesetzt. Hierbei ist jedoch zu bedenken,daß die 1. Zeile 0 ist. Es ist einfacher zu handhaben,als der CURSOR Befehl,weil nur ein Parameter übergeben werden muß.

0056	86	Wenn mittels CONSOLE Parameter übergeben wurden,dann steht in dieser Adresse die erste Zeile des Scroll-Bereiches. z.B. CONSOLE 3,16,6,31. Adresse 86 würde dann 3 enthalten.
0057	87	Beinhaltet die letzte Zeile des Scroll-Bereiches.Nicht die Anzahl der Zeilen.Im Beispiel wären es 18 (1.Zeile = 0)
0059	89	Mit Hilfe dieser Adr. kann bei GET ein Autorepeat erzeugt werden.Die Taste kann dann ständig gedrückt bleiben und muß nicht ständig neu gedrückt werden.Dieses wird durch POKE89,240. POKE89,83 ist die Ausgangsstellung.
005B	91	Beinhaltet die Anfangsspalte des Scroll-Bereiches.In dem obigen Beispiel wäre es 6.
005C	92	Hier wird die Endspalte des Scroll-Bereiches abgelegt. Im Beispiel wäre es also 37. (6+31)
005D	93	Farbwertinformationen werden hier abgespeichert.Nach dem Laden des Basics steht hier der Wert 113 (71H). 7 = Weiß 1 = Blau . POKE93,$27 würde die Vordergrundfarbe auf Rot und Hintergrundfarbe auf Weiß ändern. Wird hierzu noch 128 zugerechnet,dann erfolgt die Ausgabe im zweiten Zeichensatz.Z.B. POKE93,$27+128:PRINT"BBG BUCH"
005F	95	Diese Adresse beinhaltet den ASCII Wert der letzten gedrückten Taste.
0060	96	Speicher für das Zeichen,welches den Cursor darstellt. Dieses Zeichen kann anhand der Anzeigekode Liste verändert werden.Die Grundeinstellung ist EFH (239) im Alpha Modus und FFH (255) im Graph Modus.Dieses Zeichen kann nun mit POKE96,73 z.B. verändert werden 73 entspricht '?'. Bei jeder Modusumschaltung wird das Zeichen jedoch wieder auf den alten Wert zurückgesetzt.
0064	100	AM/PM Umschaltung wenn die Zeit TI$ unter 12.00 Uhr AM ist, dann enthält die Zelle '0' andernfalls '1'. POKE100,0 würde die Uhr von 164312 auf 044312 setzen.

0124	292	Bei POKE292,1 wird bei jedem Druck von 'CR' ein kurzer Ton erzeugt. POKE292,4 schaltet diese Option wieder ab.
018F	399	POKE399,150 erzeugt einen Ton wenn eine Taste gedrückt außer 'CTRL','SHIFT' und 'CR' gedrückt wird. POKE399,153 schaltet wieder zurück.Diese Möglichkeit ist vor allen bei Arbeieten ohne Bildschirm nützlich.Bei den Definitionstasten 'F1' bis 'F5' wird bei jedem Buchstaben ein Ton erzeugt.
0288	648	Hier kann die Geschwindigkeit manipuliert werden,mit der sich der Cursor bei der Dauertastenabfrage bewegt.Der Grundwert beträgt 10.Bei Werten über 10 wird der Cursor langsamer und bis zu 1 wird er schneller.
04B3	1203	Wird während der Programmausführung die 'BREAK' Taste gedrückt,wird die Programmausführung für kurze Zeit unterbrochen.Dieses kann durch POKE1203,201 unterbunden werden. Die Grundwert der Speicherzelle ist 216.
0655	1621	Aufwärtsscrolling des Bildschirms.USR(1621) hat den selben Effekt,wie das Drücken der 'SHIFT' und 'Cursor hoch' Taste.
067D	1661	Abwärtsscrollen des Bildschirms.Zu Bedienen wie vorherige Routine.Scrollrichtung hier jedoch abwärts.
07F9	2041	Mit dieser Zelle kann das Aussehen des Alpha Modus Cursors verändert werden.Der Ausangswert ist 239.POKE2041,106 würde den Cursor in '+' verändern.Die Änderung wird jedoch hier erst nach der Rückkehr aus dem Graph Modus ausgeführt.
0A32	2610	Beinhaltet den Tempo Wert für Musik.Wenn das Tempo 1 ist, steht hier 7,bei Tempo 6 steht hier 2 usw. Normal ist 4
0A39	1617	Die Bytes in 1617 und 1618 kontollieren den Tongenerator. Der Tongenarator wird mit USR(68) gestartet und mit USR(71) wieder abgeschaltet.

0FFC	4092	Hier steht der Filecode des letzten geladenen Programms. 5 bedeutet MZ-700 Basicprogramm 4 Daten 2 MZ-80K Basic Programm
0FFD	4093	Ab dieser Adresse steht der Name des letzten Programms,das von der Kassette geladen wurde.Da ein Programmname maximal 16 Zeichen enthalten kann geht der Bereich bis 4109 (100DH) Um den Programmnamen zu ermitteln gebe man ein: FORA=4093TO4109:?CHR$(PEEK(A);:NEXT
1010	4112	In dieser und der folgenden Speicherzelle wird die Länge letzten geladenen Programmes abgelegt.Da die Länge auch größer als 255 sein kann,muß die Adresse 4113 (1011H) auch mitbenutzt werden.Die Länge des Programms kann mit Basic wie folgt berechnet werden: PRINTPEEK(4110)+PEEK(4111)*255
1010	4112	In 4112 und 4113 steht die Adresse wohin das Programm geladen wurde.Im S-Basic ist dieses 27599 (6BCFH).

Nun wird noch einmal eine Auflistung der wichtigsten Routinen gegeben,die im Basic Monitor,im nachfolgenden auch RAM Monitor genannt,vorhanden sind. Dieser Monitor ist weitgehend kompatibel zum ROM Monitor,welcher gleich nach dem Einschalten des Gerätes zur Verfügung steht.
Die Sprungliste am Anfang bis 0066H ist mit der im ROM Monitor identisch.
Hier nun eine übersicht über die Routinen von 0000H bis 0066H.
Alle Angaben sind Hexadezimal zu verstehen.Bei 'Bedeutung' steht der Name, welcher auch im Handbuch von Sharp angegeben wurde.

Adr.	Inhalt	Mnemonic	Bedeutung
0000	C3DA00	JP 00DA	Start und 1819H wird mit 0 geladen.
0003	C32001	JP 0120	Get Line. Nicht benutzt (siehe 004A)
0006	C3FE04	JP 04FE	New Line 2
0009	C3FA04	JP 04FA	New Line wenn Cursor nicht bereits in einer neuen Zeile ist.

000C	C33105	JP 0531	Print Space
000F	C30205	JP 0502	Print Tab auf Position 0,10,20 oder 30
0012	C33305	JP 0533	Print Character des A Registers im Ascii
0015	C3DA04	JP 04DA	Print Message. Nicht benutzt (siehe 0051)
0018	C3DA04	JP 04DA	Print Message. Nicht benutzt (siehe 0051)
001B	C3F002	JP 02F0	Get Key. Nicht benutzt
001E	C3A004	JP 04A0	Break
0021	C33B0A	JP 0A3B	Write Information (Header)
0024	C33F0A	JP 0A3F	Write Data (Programm)
0027	C37A0A	JP 0A7A	Read Information (Header)
002A	C37E0A	JP 0A7E	Read Data (Programmm)
002D	C3940A	JP 0A94	Verify
0030	C3E408	JP 08E4	Melody ab 'DE' bis 0DH
0033	C35A0D	JP 0D5A	Set Time
0036	0000	NOP,NOP	keine Befehle
0038	C3E40D	JP 0DE4	Interuptroutine
003B	C3A70D	JP 0DA7	Read Time Register A enthält 0 für AM oder 1 für PM und 'DE' die Sekunden.
003E	C3130A	JP 0A13	Beep kurzer Ton
0041	C3040A	JP 0A04	Set tempo Register A von 1 bis 7
0044	C3B709	JP 09B7	Melody Start

0047	C3D209	JP 09D2	Melody Stop
004A	C32001	JP 0120	Get Line (siehe 0003)
004D	00	-	Plot Flag 1=an,2=aus
004E	00	-	Console flag 1=an,2=aus
004F	FF	-	Basic Flag FF=BasIc,64=Monitor
0050	00	-	Benutzt bei Aufzeichnungen auf Kassette
0051	C3DA04	JP 04DA	Print Message
0054	00	-	Cursor Spalte
0055	00	-	Cursor Zeile
0056	00	-	Console Start Zeile
0057	18	-	Console Anzahl der Zeilen
0058	C35302	JP 0253	Get Key Tastenabfrage ohne 'Repeat'
005B	00	-	Console Start Spalte
005C	27	-	Console Anzahle der Spalten
005D	71	-	Bildschirmfarbe. 71 = Weiß auf Blau
005E	00		
005F	00	-	Ascii Wert der letzten gedrückten Taste
0060	EF	-	Anzeigecode des jetzigen Cursors
0061	C36A0E	JP 0E6A	Joystick Routine
0064	00	-	AM/PM Flag. 0=AM,1=PM

0065	00	-	Für Joystick
0066	00	-	Für Joystick

Des weiteren sollen hier auch noch die Adressen angegeben werden,wo die einzelnen Routinen beginnen,die im RAM-Monitor vohanden sind.
Es werden jeweils die Einsprungadressen in hexadezimaler Schreibweise angegeben.

1443 SAVE	1469 LOAD	14A9 VERIFY
14B8 RETURN	1545 GOTO	154A DUMP
15BD MEMORY SET	15F4 FIND	165A TRANSFER

1.4 Nützliche Adressen im Basic

Die nachfolgende Liste des Basic-Monitors und des Basics gibt eine übersicht über nützliche Adressen und Unterprogramme.

0067 bis 00A6 liegen die 'CTRL' Adressen.Diese bestimmen die Sprungadressen nach dem ?CHR$(Befehl oder der CTRL Taste zusammen mit einer anderen Taste.

00EA Diese Routine wird von PEEK aufgerufen,um auf den Video-RAM zuzugreifen.

00F2 Diese Routine wird gebraucht um in das Video-RAM mit POKE zuzugreifen.

04A0 Diese Routine überprüft,ob die BREAK oder SHIFT/BREAK Taste gedrückt wurde.

04CD Hier werden ASCII Kode Zeichen in den Anzeigekode (Display code) umgesetzt.

0EFC Wie vorherige Routine,jedoch wird hier genau umgekehrt verfahren. (Display code in ASCII code)

0FFC In dieser Adresse wird der 'Filecode des letzten geladenen Prgs. abgespeichert.

0FFD Ab dieser Adresse wird der Name des letzten geladenen Programms abgelegt. Maximal 16 Zeichen

100E Hier steht die Länge des letzten geleadenen Programms.An erster Stelle das niederwertige Byte und dann das höherwertige.

1010 Die Selbststartadresse des Programms steht an dieser Stelle.Bei Basic jedoch ohne Bedeutung.

110F Buffer.Hier steht die gesamte letzte Zeile,die über die Tastatur eingegeben wurde.

1322 Ab hier stehen die Befehle,mit denen die Funktionstasten belegt F1-F10 belegt sind.Die Tabelle reicht bis 13C1.

1822 Hierhin springt das Basic wenn es geladen wurde.Diese Routine löscht auch den Bereich bis FF00.Wenn z.B. in Adresse 182E nun A0 geschrieben wird,wird der Speicher nur bis A000 gelöscht.
Hier steht auch der Meldetext nach dem Start.

1848 Hierhin gelangen alle Rücksprünge vom ROM-Monitor ins Basic. (Ram Test)

1869 Dieses ist die Adresse von welcher aus die Ausgabe 'Ready' auf den Bildschirm erfolgt.

1872 Ab dieser Adresse liegt die Hauptroutine für Eingabe und die Veränderung von Basiczeilen.

18FE Hier wird die Ausführungsroutine gestartet (RUN).Außerdem wird hier auch überprüft,ob TRON oder TROFF Modus besteht bei Adresse 1919 und bei 1930 wird die BREAK Taste noch einmal überprüft.

19B9 Ab dieser Adresse liegt die Routine,welche MZ-80K Programme in MZ-700 umsetzt.

20C7 Hier sind die ERROR Tabellen und die jeweils dazugehörigen Nummern abgelegt.

21A7 Diese Adresse beinhaltet die derzeitig aktuelle Zeilennummer der Basiczeile.

2287 Ab dieser Adresse liegt die im Basichandbuch nicht aufgeführte TRON Routine.

242F Von hier bis 245B stehen Bytekombinationen,welche nicht benutzt werden.

2AF5 Von dieser Adresse bis zu Adresse 2CA8 steht die erste 'Keyword' Tabelle.

2CA9 Von hier bis 30A8 ist ungenutzter Speicher.hier können z.B. eigene Zusatzroutinen implementiert werden.

30A9 Ab hier steht die zweite 'Keyword' Tabelle.Sie reicht bis an die darauffolgende Adressen Tabelle.

3147 Von hier bis 3304 stehen die Adressen der einzelnen 'Keywords'. Sie sind in der Reihenfolge analog zu den 'Keyword' Tabellen aufgegebaut.

3807 Ist die Adresse von GOTO.Wenn nach dem GOTO oder GOSUB Token OC steht erfolgt im S-Basic immer gleich die Absolutadresse im Speicher.Bei OB folgt die Zeilennummer.Dieses hat eine schnellere Ausführung des Programms zur Folge.

38D3 Bis 38EF sind Zeiger und Werte von dem aktuellen Programm abgelegt. Die Aufgaben folgen in den nächsten Zeilen.

38D5 Stack Pointer.

38D8 GOSUB Flag.

38D9 RUN Flag. 1=Stop 0=Run

38DD TRON Flag. TROFF=0 TRON=1

38E6 sowie die nachfolgende Adresse 38E7 beinhalten die derzeitige Zeilennummer.

3D86 Von hier ab liegt die Umrechnungsroutine von Binär nach Dezimal im im Speicher.

4795 Ab dieser Adresse liegt die Routine,welche eine Basic Zeile in Token umsetzt.In dieser Token Schreibweise wird das Programm auch im Speicher des Computers abgelegt.

49CD Diese Routine bewirkt genau das Gegenteil,wie die Routine 4795,jedoch werden die Token hier wieder für die Bildschirmausgabe umgesetzt.

4B72 bis 4D4F sind die Fehler Mitteilungen abgelegt.Diese Mitteilungen stehen im Klartext (ASCII) im Speicher.

4D50 Kontrolle für Integer Variablen deren Wert maximal 255 ereichen kann (wie z.B. bei CHR$)

4D65 Kontrolle für Integer Variablen deren Wert über 255 liegen kann (wie z.B. MOVE)

4DCB Diese Speicherzelle dient als Plotter Modus Steuerung und Anzeige. Wenn der Wert 1 ist,dann ist der Plotter im Text Modus.Bei 2 ist der Plotter auf Graphik Modus gestellt.

4DCD Diese Adresse gibt die Anzahl de Zeichen pro Zeile im Text Modus an.Die Inhalte haben folgende Bedeutung:
78 = Normal (40 Zeichen pro Zeile)
76 = Groß (26 Zeichen pro Zeile)
83 = Klein (80 Zeichen pro Zeile)

528B Bei dieser Routine werden alle mathematischen Ausdrücke auf ihre Richtigkeit überprüft.

6AB3 bis 6AC1 liegen weitere Zeiger und Merkzellen für das aktuelle Basicprogramm.

6AB3 Ende des Basicprogramms

6AB5 Variablenzeiger

6AB7 Ende des gesamten Basicprogramms mit Variablen

6AB9 'Stack pointer' für Basic

6ABD Ende des verfügbaren Speichers

6ABF Beginn des Basicprogramms

1.5 Farbspeicher und 2. Zeichensatz

Bei der Darstellung von Zeichen und Farben auf dem Bildschirm ist immer folgendes zu bedenken.
Es gibt zum einen den Zeichenspeicher.Dieser Zeichenspeicher liegt im Speicherbereich von D000H bis D3E7H (53248 bis 54247).Dieses sind genau 25 Zeilen mit jeweils 40 Zeichen.In diesem Bereich werden nur die Zeichen erzeugt.
Die dazugehörigen Farbinformationen liegen genau 2048 Byte höher.
Also im Bereich von D800H bis DBE7H (55296 bis 56295).
Nun ein kleines Beispiel wie ein Zeichen mit Farbe erzeugt werden kann.
Das Zeichen erscheint in der oberen linken Ecke.

```
10 POKE 53248,83 : REM Zeichen erzeugen
20 POKE 56295,$40 : REM Farbe erzeugen hier Grün auf Schwarz
```

Der 2. Zeichensatz wird immer dann aktiviert,wenn der Farbspeicher mit einem Wert geladen wird,der größer als 128 (80H) ist.
Deshalb muß zu der Farbe nur noch 128 zugerechnet werden.
Z.B. wird der ganze Bildschirminhalt auf den zweiten Zeichensatz umgeschaltet und die Farbe auf Grün/Schwarz gesetzt durch:

```
FORX=55296TO56296:POKEX,$40+128:NEXT
```

Aber auch PRINT Anweisungen können einfach im 2. Zeichensatz dargestellt werden.Hierzu dient die Speicherstelle 93 (5DH).
Auch hierzu ein kleines Beispiel:

```
POKE$5D,PEEK($5D)+128:"700 BEISPIEL"
```

1.6 Konvertieren von MZ-80K Programmen

Das S-Basic hat zwar eine Routine,die MZ-80K Programme für den MZ-700 umsetzt,aber diese Routine kann nur die Token umsetzen.
Vor allen PEEK's,POKE's und USR's sollten vor dem Starten des Programms noch einmal überprüft werden.Alle POKE's zwischen 53248 und 54247 greifen auf den Videobereich zu sind weiterhin kompatibel.
Die anderen POKE's beeinflußen das Basic meist direkt,daher sollten sie mit Vorsicht benutzt werden.
Hier folgt jetzt eine kurze Liste von POKE's,die auf jeden Fall geändert werden sollten.
4465 und 4466 sind die Adressen,die beim MZ-80K die Steuerung des Cursors erledigen.Eine Zeile könnte z.B. wie folgt aussehen:
POKE4465,6:POKE4466,15
4465 gibt die Spalte an.4466 gibt die Zeile an.
Dieses kann nun jedoch recht einfach verändert werden.
POKE84,6,15   oder   CURSOR6,15
17828 ist beim MZ-80K die Adresse für Dauertastenabfrage.Dieses muß mit POKE89,240 vorher abgeändert werden.Wird im Programm dann irgendwo etwas wie IFPEEK(17828)=65THEN... dann muß dieses auch noch in IA$="A"THEN... geändert werden.
4513 und 4514 sind beim MZ-80K für die Steuerung des Tongenerators verantwortlich.Diese Adressen sind beim MZ-700 durch 2617 und 2618 realisiert.
POKE4464,1 schaltet beim 80K die Tastatur auf Kleinschrift.Dieses wird geändert in PRINTCHR$(5).POKE4464,0 schaltet beim 80K wieder auf Großbuchstaben dieses wird beim MZ-700 mit PRINTCHR$(6) durchgeführt.
Dieses sind im Prinzip die wichtigsten POKE's die geändert werden müssen.
Bei allen anderen POKE's sollte mit extremer Vorsicht vorgegeangen werden.
Vorsichtshalber sollten unbekannte POKE's erst einmal ins REM gelegt werden.
Adressen wie 10167,10680,10682,7125 sollten sie sowieso ersatzlos löschen.

1.7 Logische Operatoren AND/OR

In vielen Zeitschriften und Programmlistings sind die Befehle AND/OR angegeben.über diese Befehlswörter verfügt das S-Basic jedoch nicht.Aber das sollte keine Probleme darstellen,weil dieses sehr leicht ersetzt werden kann durch die Zeichen '*' und '+'.
Das '+' Zeichen steht für OR
Das '*' Zeichen steht für AND
Beispiel 1) für 'IFA=1ANDB=2THEN..' geht 'IF(A=1)*(B=2)THEN..'
2) für 'IFA=1ORB=2THEN...' geht 'IF(A=1)+(B=2)THEN...'
Es ist jedoch wichtig,daß alle Argumente die so benutzt werden in Klammern zu stehen haben.

1.8 BREAK und SHIFT/BREAK Tasten sperren

Hier wird nun angegeben wie ein Programm gegen das Drücken der BREAK und der SHIFT/BREAK Taste abgesichert werden kann.
Das Drücken der BREAK Taste alleine unterbricht ein laufendes Programm nicht,sondern hält es nur für kurze Zeit an.Dieses kann jedoch durch: POKE1203,201 verhindert werden.POKE1203,216 ist die Normalstellung.
Zum Verhindern von SHIFT/BREAK gilt es mehrere Speicherzellen zu verändern.
Dazu folgende Zeile eingeben:POKE6452,54,25
POKE6452,113,32 ist wieder normal.
Nun kann jedoch auch noch bei INPUT das Programm unterbrochen werden.Um dieses zu verhindern dient die nächste Zeile.
POKE8987,29,35:POKE9056,98,35
Und um diese Option wieder abzuschalten gebe man ein:
POKE8987,105,32:POKE9056,105,32

1.9 Farbwechsel ohne Bildschirm löschen

Normalerweise gibt es im S-Basic keine Möglichkeit dem gesamten Bildschirm ,über den COLOR Befehl,eine neue Farbe zuzuordnen,ohne das dabei der Bildschirminhalt gelöscht wird.
Hier wird jetzt eine Möglichkeit vorgestellt,wie sowohl die Vordergrund-, als auch die Hintergrundfarbe des gesamten Bildschirms geändert werden kann,ohne den Inhalt zu löschen.
Dieses kann vom Programmierer dann sehr effektvoll in den Programmen eingesetzt werden.

```
COLOR,,6,1:USR($072D)
```

Mit dem obigen Befehl wird der Farbwechsel durchgeführt.
Es ist jedoch zu beachten,daß der Cursor nach Ausführung von dem USR immer wieder in die HOME Position (oben,links) geht.

1.10 Abgeleitete Funktionen

Das S-Basic hat bereits eine Vielzahl von mathematischen Funktionen,aber viele Funktionen sind im Befehlssatz des Basics nicht vorhanden,die vor allen bei mathematischen Programmen sehr nützlich sein können.
Diese Funktionen können aber aus den vorhandenen Befehlen abgeleitet werden.
Daher wird hier eine Aufstellung von weiteren wichtigen und gebräuchlichen Funktionen gegeben.

Sekans	SEC(X)=1/COS(X)
Cosekans	CSC(X)=1/SIN(X)
Cotangens	COT(X)=1/TAN(X)
Arcussinus	ARCSIN(X)=ATN(X/SQR(-X*X+1))
Arcuscosinus	ARCCOS(X)=-ATN(X/SQR(-X*X+1))+π/2
Arcussekans	ARCSEC(X)=ATN((SQR(X*X-1))+(SGN(X)-1)*π/2

Arcuscosekans	ARCCSC(X)=ATN(1/SQR(X*X-1))+(SGN(X)-1)*π/2
Arcuscotangens	ARCCOT(X)=-ATN(X)+π/2
Hyperbelsinus	SINH(X)=(EXP(X)-EXP(-X))/2
Hyperbelcosinus	COSH(X)=(EXP(X)+EXP(-X))/2
Hyperbeltangens	TANH(X)=-EXP(-X)/(EXP(X)+EXP(-X))*2+1
Hyperbelsekans	SECH(X)=2/(EXP(X)+EXP(-X))
Hyperbelcosekans	CSCH(X)=2/EXP(X)-EXP(-X))
Hyperbelcotangens	COTH(X)=EXP(-X)/(EXP(X)-EXP(-X))*2+1
Areasinus	ARSINH(X)=LN(X+SQR(X*X+1))
Areacosinus	ARCOSH(X)=LN(X+SQR(X*X-1))
Areatangens	ARTANH(X)=LN((1+X)/(1-X))/2
Areasekans	ARSECH(X)=LN((SQR(-X*X+1)+1)/X)
Areacosekans	ARCSCH(X)=LN((SGN(X)*SQR(X*X+1)+1)/X)
Areacotangens	ARCOTH(X)=LN((X+1)/(X-1))/2

1.11 Anlegen einer Sicherheitskopie vom S-Basic

Es ist zu empfehlen sich von dem S-Basic eine Kopie anzufertigen und diese Kopie dann auch im alltäglichen Gebrauch zu benutzen,da die Original-Cassette ja durch einen Zufall auch überspielt oder gelöscht werden könnte.
Um diese Kopie anzufertigen gehe man wie folgt vor:
- Computer anschalten
- 'MC000' und dann 'CR' eingeben
- 'CD2700CD2A00C30811' und 'CR' eingeben
- SHIFT/BREAK drücken
- Nun Original S-Basic Cassette einlegen (Rückgespult)
- 'JC000' und 'CR' eingeben
- Cassettenrekorder Taste 'PLAY' drücken
- Warten bis 'HIT ANY KEY' erscheint
- Original Cassette entnehmen
- Leercassette (Rückgespult) einlegen
- Eine Taste drücken
- Cassettenrekorder Tasten 'RECORD' und 'PLAY' drücken
- etwa 6 1/2 Minuten warten
- Die Cassette entnehmen.Das Basic ist nun auf dieser Cassette gespeichert.

Dieses Kopieren funktioniert immer dann,wenn das Basic noch nicht gestartet wurde.

1.12 Erweiterung des Befehlssates vom S-Basic

Hier werden noch einige Erweiterungen für das S-Basic vorgestellt.
Alle folgenden Erweiterungen benötigen keinen weiteren Speicherplatz,weil Sie noch freien Speicher innerhalb des Basics benutzen.
Zum Eingeben der Zusatzroutinen und Erweiterungen laden Sie zunächst das S-Basic von Cassette und geben dann alle Zeilen an vor denen Zeilennummern stehen.Die Texte davor und dahinter sind lediglich Kommentare.

1) Ready mit Ton

Die folgende Routine erzeugt bei jedem 'Ready' einen kurzen Ton. Dieses ermöglicht es,daß nach LOAD oder anderen längeren Vorgängen deren Ende auch akustisch wahrgenommen werden kann.

```
100 REM ** Ready Ton **
110 POKE6256,129,48:POKE12417,175,205,62,0,205,6,0,201
```

2) Musik bei 'Error'

Bei jedem Error wird durch diese Routine ein kurzer tiefer Ton erzeugt. Es wird also dadurch ein Fehler nicht nur auf dem Bildschirm angezeigt, sondern auch mit einem tiefen Ton begleitet.

```
120 REM ** Musik bei 'Error' **
130 POKE8561,137,48
140 POKE12425,205,81,0,229,17,149,48,205,48,0,225,201,45,65,13
```

3) LIST anhalten mit 'SPACE'

Diese Routine ermöglicht es beim auflisten von Programmen diese mit 'SPACE' anzuhalten um sich z.B. mit beiden Händen Aufzeichnungen zu machen.Bei einem weiteren Druck auf 'SPACE' wird Zeile für Zeile weitergelistet.Bei drücken einer beliebigen anderen Taste geht das Listing normal weiter.

```
150 REM ** LIST anhalten **
160 POKE16786,77,36,204,83,2,0:POKE9293,205,27,0,254,32,201
```

4) Verbesserter TRON Befehl

Bei diesem verbesserten 'TRON',wird die Zeilenzahl,welche gerade abgearbeitet wird,in der oberen linken Ecke angezeigt,anstatt immer an der aktuellen Cursorposition.Dieses erleichtert die übersicht.Der 'TROFF' Befehl schaltet dieses neue 'TRON' wieder ab.

```
170 REM ** TRON neu **
180 POKE8841,42,230,56,205,167,33,33,0,208,6,5,26,205,205,4,205,242,0
190 POKE8859,35,19,16,245,225,241,201
```

5) ROPEN und WOPEN Anzeige

Bei den Vorgängermodellen des Sharp MZ-700,war bei WOPEN und ROPEN eine Aufforderung für 'PLAY' oder 'RECORD PLAY' implementiert.
Im S-Basic funktioniert diese Anzeige jedoch nur im Direkt Modus und nicht im Programm selbst.
Es läßt sich jedoch leicht auch im Programm realisieren.Dazu dient die folgende Zeile.Sie gibt 'PLAY' bzw. 'RECORD PLAY' bei ROPEN bzw. WOPEN aus.

```
200 REM ** PLAY/RECORD PLAY bei ROPEN/WOPEN **
210 POKE18242,205:POKE18252,175
```

Jetzt wird aber immer noch nicht angezeigt,ob die Daten geladen werden.
Die folgende Routine schreibt vor dem Einladen den Namen der Daten auf den Bildschirm.

```
220 REM ** ROPEN mit Namensanzeige **
230 POKE18167,205,153,48
240 POKE12441,17,11,67,33,252,15,205,200,22,58,252,15,201
```

6) Spezial LIST

Diese Abänderung der LIST Routine bewirkt,daß nach jeder Basiczeile eine Leerzeile ausgegeben wird.Dieses erleichtert den überblick besonders bei längere Programme.Diese Erweiterung gilt sowohl für den Bildschirm,als auch für den Drucker.

```
250 REM ** Spezial LIST (mit Leerzeile) **
260 POKE12410,205,249,23,205,249,23,201
270 REM** an **:POKE16781,122,48
280 REM** aus **:POKE16781,249,23
```

Mit dem POKE in Zeile 270 wird das Spezial List aktiviert.
Der POKE in Zeile 280 schaltet das Spezial List wieder ab.

7) Schützen von Programmen

Das Schützen von Programmen gegen LIST oder SAVE kann nur ereicht werden,wenn das Programm bereits mit RUN einmal gestartet wurde.
Ein Programm kann also nicht 100% geschützt werden.
Hier soll jedoch trotzdem gezeigt werden,wie ein Programm gegen verschieden Befehle gesichert werden kann.
POKE16642,201 verhindert LIST.POKE16642,175 ist Normal.
POKE17060,201 Verhindert SAVE.POKE17060,205 ist Normal
POKE26037,201 verhindert PEEK.POKE26037,254 ist Normal
POKE 5058,201 verhindert BYE. POKE 5058,229 ist Normal

8) Laden des geänderten Basics

Die eben angeführten Zeilen sollten in den Computer eingegeben werden.
Nun hat man ein Basicprogramm,welches wenn es gestartet wurde die Veränderungen erzeugt.Der Benutzer sollte sich lediglich die Adressen des POKE Befehls merken,die in den Zeilen 270 und 280 stehen,damit er das Spezial List ein- und ausschalten kann.Nach dem starten des Programms ist das Spezial List noch ausgeschaltet.
Diese Erweiterungsprogramm sollte dann direkt nach dem S-BASIC auf die Cassette 'gesaved' werden.Es kann dann immer bei Bedarf zugeladen werden.
In Zeile 300 kann ja dann noch eine Meldung geschrieben werden,die etwa so so aussehen könnte:
CLS:?"Basic Erweiterung eingefügt":?SIZE;" Byte freier Speicher"

DER MONITOR 1Z-013A

Der MONITOR 1Z-013A ist das allgemeine Betriebssystem des MZ-700.Sofort nach einschalten des Computers meldet sich der Monitor und erwartet eine Befehlseingabe.
Der Monitor liegt im Bereich 0000H bis 0FFFH und wird permanent in dem sogenannten M-ROM gespeichert.Dieses M-ROM ist ein EPROM (elektrisch programmierbarer Festwertspeicher) und befindet sich in einem IC-Sockel auf der Hauptplatine des Computers.Dadurch hat man die Möglichkeit diesen IC zum Beispiel auszutauschen und durch einen kompatiblen aber Befehlserweiterten neuen Monitor,wie den BBG-Monitor,der eine Komplette Maschinensprache enthält,zu ersetzen.
Bei Bedarf kann man den Monitorbereich (0000H-0FFFH) aber auch in einen RAM-Bereich legen.Siehe hierzu Kapitel Bank-Switching.Das von der Firma Sharp mitgelieferte S-Basic nutzt diese Möglichkeit.Das S-Basic hat einen eigenen Monitor.Die Befehle dieses Monitors sind ebenfalls im Sharp-Handbuch beschrieben.
Nach Einschalten des Computers oder nach betätigen der Reset-Taste meldet sich der MONITOR 1Z-013A und erwartet eine Befehleingabe.Die Monitor-Befehle sind im Sharp-Handbuch (S 145-152) ausführlich beschrieben.Ein Befehl wurde jedoch im Handbuch vergessen:

D-Befehl: Dieser Befehl bewirkt eine Hexadezimale Auflistung des Speichers mit ASCII-Ausgabe.
DXXXX bewirkt die Auflistung ab der hexadezimalen Adresse XXXX.
Die Auflistung wird beendet,wenn eine Bildschirmseite beschrieben ist.
DXXXXYYYY bewirkt die Auflistung von XXXX bis YYYY.

Im Monitor 1Z-013A sind jedoch nicht nur Grundbefehle implementiert.Es sind alle wichtigen Subroutinen,wie Tastaturabfrage oder Bildschirmausgabe vorhanden.Diese Subroutinen und ihre Funktionsweise sollen im folgenden beschrieben werden.
Da hierfür gewisse Grundkenntnisse der Maschinenprogrammierung das Verstehen erleichtern,wäre es bei Computerneulingen eventuell angebracht erst einmal den Maschinensprachekurs zu lesen.

Der erste Teil des Monitors besteht aus einer Sprungtabelle.Die Sprungtabelle erfüllt zwei wesentliche Funktionen:

a)Sie ermöglicht die Kompatibilität zu den Vorgängermodellen MZ-80K und MZ-80A.

B)Die wichtigen Monitorroutinen lassen sich einfacher merken.

Die Monitorroutinen werden mit dem Maschinensprachebefehl CALL(XXXXH) angesprungen.Nach Ausführung der entsprechenden Monitorroutine wird automatisch zur Adresse,die die Monitorroutine aufgerufen hat,zurückgesprungen (wie der Befehl 'GOSUB' im S-Basic).

MONITOR UNTERROUTINEN

0000H Start des Monitors

Diese Adresse ist keine Subroutine und darf nur mit einem Sprung aufgerufen werden.Der Aufruf dieser Adresse erfolgt meistens nur nach Beendigung eines Programmes.

0003H GET LINE

GET LINE ermöglicht die Eingsbe einer Zeile,also mehreren Zeichen von der Tastatur.Die eingegebene Zeichenkette wird im Speicher an der durch das Register DE bestimmten Stelle abgelegt.Die Routine endet nach drücken der 'CR' Taste.Nach Beendigung der Subroutine wird eine Kennung (0DH) an der Zeichenkette angefügt.Diese Kennung ist notwendig um des Ende der eingegebenen Zeichenkette zu markieren.
Neben dem Eingeben der Zeichenkette wird diese gleichzeitig auf dem Bildschirm angezeigt..
Es können maximal 80 Zeichen eingegeben werden.
Bei Betätigen der SHIFT&BREAK Taste wird anstatt der Zeichenkette der Break-Code an der Adresse,die im DE-Register steht,abgespeichert.

0006H NEWLINE

Der Cursor wird an den Beginn der nächsten Zeile gesetzt.

0009H NEWLINE

Der Cursor wird an den Beginn der nächsten Zeile gesetzt,falls er nicht schon am Anfang einer Zeile positioniert ist.

000CH PRINT SPACE

Druckt an der momentanen Cursorposition ein Leerzeichen.

000FH PRINT 10 SPACES
Druckt ab der momentanen Cursorposition maximal 10 Leerzeichen bis zur nächsten TAB-Position (0,10,20,30).

0012H PRINT CHARACTER
Das im Akkumulator gespeicherte ASCII-Zeichen wird ausgegeben.Die Steuerzeichen ODH und 11-16H erzeugen keine Zeichen,ihre Funktionen werden jedoch ausgeführt,Wenn man den Akku z.B. mit 16H lädt und diese Routine aufruft,wird der Bildschirm gelöscht (CLEAR-HOME).

0015H PRINT MESSAGE
Ausgabe einer Zeichenkette beginnend mit der Cursorposition auf dem Bildschirm.Die Anfangsadresse der Zeichenkette wird im DE-Register übergeben.Die Zeichenkette muß mit einem ODH abgeschlossen werden. Die Steuerzeichen 11H-16H werden ausgeführt.

0018H PRINT MESSAGE
Ausgabe einer Zeichenkette beginnend mit der Cursorposition auf dem Bildschirm.Die Anfangsadresse der Zeichenkette wird im DE-Register übergeben.Die Zeichenkette muß mit einem ODH abgeschlossen sein. Die Steuerzeichen 11H-16H werden nicht ausgeführt,sondern invers ausgegeben.

001BH GET KEY
Ein Zeichen im ASCII-CODE wird von der Tastatur gelesen und der Wert im Akku gespeichert.Ist keine Taste gedrückt,enthält der Akku bei der Rückkehr den Wert 00H.
Sondertasten liefern den entsprechenden ASCII-Wert.

001EH SHIFT,BREAK,CONTROL
Wenn die BREAK-Taste gedrückt wurde,ist das CARRY-FLAG auf "1" gesetzt.
Wenn die SHIFT-Taste gedrückt wurde,ist Bit 6 im Akku auf "1" gesetzt.
Beim drücken der CONTROL-Taste ist das Bit 5 im Akku "1" und bei gleichzeitigem betätigen der Tasten SHIFT&CONTROL ist Bit 4 im Akku auf "1" gesetzt.

0021H WRITE HEADER
Der Programmkopf (Header) wird auf Cassette geschrieben.Der Header enthält folgende Daten:Programmname,Programmanfang,Programmlänge Filecode und die Startadresse.Die genannten Daten werden in den Adressen 11F0H-116FH abgelegt.Siehe auch Monitor-Hilfszellenbelegung.

0024H WRITE DATA
Das durch die Anfangsadresse und durch die Länge gekennzeichnete Programm (Datenblock) wird auf Cassette abgespeichert.Beim Original Sharp-Monitor wird dieser Datenblock doppelt abgespeichert.Im BBG-Monitor benötigt das Saven nur noch halb soviel Zeit,weil das Programm nur noch einfach abgespeichert wird.

0027H READ HEADER
Der Programmkopf wird von Cassette eingelesen.Der Programmkopf enthält den Programmnamen,den Programmanfang,die Länge,den Filecode und die Startadresse.

002AH READ DATA
Lädt ein Programm,das durch die Anfangsadresse und durch die Länge lokalisiert wird,in den Speicher.

002DH VERIFY
Das auf Cassette befindliche Programm wird mit dem im Speicher befindlichen Programm verglichen.Bei einem Vergleichsfehler wird eine Fehlermeldung ausgegeben.

0030H MELODY
Spielt Musik entsprechend einer Zeichenkette,deren Anfangsadresse im DE-Register gespeichert ist.Die Zeichenkette muß mit einem 0DH oder einem C8H abgeschlossen sein.Zusätzlich ist in dieser Routine eine BREAK-Abfrage eingebaut.Beim Drücken der BREAK-Taste wird das CARRY-FLAG gesetzt.

0033H SET TIME
Stellen und starten der eingebauten Uhr.Der Akku wird mit 0 oder 1 (AM oder PM) übergeben und das DE-Register wird mit der Sekundenanzahl (maximal 43200,entspricht 12 Stunden) initialisiert.

0038H INTERRUPT
Bei einem RESTART 38H Befehl verzweigt der Z-80 Prozessor Hardwaremässig zur Adresse 0038H und springt zur dort angegebenen Adresse. Da diese Adresse jedoch im Monitorbereich liegt und nicht veränderbar ist,wird automatisch zur Speicheradresse 1038H gesprungen In dieser Adresse kann man jeden beliebigen INTERRUPT-Vektor ablegen.

003BH READ TIME
Die augenblickliche Uhrzeit wird gelesen.Im DE-Register befindet sich nach Ansprung dieser Routine die Sekundenanzahl (maximal 43200 ,entspricht 12 Stunden) und der Akku ist mit 0 oder 1 für AM oder PM geladen.

003EH BELL
Erzeugt einen kurzen Ton mit der Frequenz 880 Hz.

0041H SET TEMPO
Bestimmt die Abspielgeschwindigkeit der Musik entsprechend dem Akkumulatorinhalt.Zahlenwerte von 1 bis 7 sind zugelassen.Dabei entspricht 01H langsam,04H mittel und 07H schnell.

0044H MELODY START
Erzeugt einen Dauerton mit festgelegter Frequenz.Die Frequenz errechnet sich nach folgender Gleichung:F=895kHz/NM.NM entspricht einer 2-Byte Zahl in der Speicherstelle 11A1H.Dabei ist zu beachten daß N in 11A2H und M in 11A1H abgelegt werden müssen.

0047H MELODY STOP
Beendet die mit MELODY-START begonnene Tonerzeugung.

004AH START MONITOR
wie 0000H

00F3H JUMP-BEFEHL
Beim drücken der Taste J (Jump) wird diese Routine angesprungen.

00F7H BEEP ON/OFF BEFEHL

00FFH FLOPPY DISK BEFEHL

0111H LOAD BEFEHL

0155H PLOTTER TEST BEFEHL

018FH PLOT CHARACTER
Der gespeicherte Wert im Akkumulator wird im ASCII-CODE an den Plotter übergeben und ausgedruckt.Einige ASCII-Werte sind jedoch für Plottersteuerzeichen reserviert.Die ausführliche Beschreibung Dieser Subroutine wird im Kapitel Plottersteuerung vorgenommen.

01A5H PLOT MESSAGE
Die ASCII-Werte einer Zeichenkette werden nacheinander dem Plotter übergeben und ausgedruckt.Die Position der Zeichenkette im Speicher wird im DE-Register übergeben.Die Zeichenkette muß mit einem ODH beendet werden.Die ausführliche Beschreibung wird im Kapitel Plottersteuerung vorgenommen.

01C7H MELODY
wie 0030H

02A6H INCREMENT DE
Das DE-Register wird viermal incrementiert.(Increment bedeutet +1)

02ABH MELODY START
wie 0044H

02BEH MELODY STOP
wie 0047H

02E5H SET TEMPO
wie 0041H

030BH SET TIME
wie 0033H

035BH READ TIME
wie 003BH

038DH TIME INTERRUPT
Wenn das AM-Flag gesetzt ist,wird dieses gelöscht und das PM-Flag gesetzt bzw. umgekehrt.Gleichzeitig wird der Timer mit 43200 Sek. geladen.

03B1H DISPLAY SPACE & CHARACTER
Druckt ein Leerzeichen und ein Zeichen,das durch den ASCII-Wert im Akkumulator bestimmt wird.Die Ausgabe erfolgt an der im Register HL bestimmten Video RAM Adresse.

03DAH HEX TO ASCII
Die hexadezimale Zahl in den 4 niederwertigen Bits des Akkus wird in den entsprechende ASCII-Code umgewandelt und im Akku abgelegt.

03E5H ASCII TO HEX
Wandelt eine hexadezimale Ziffer,die als ASCII-Zeichen im Akku vorliegt in Binärform um und speichert das Ergebnis in den 4 niederwertigen Bits des Akkus.

03F9H ASCII TO HEX
wie 03E5H

0410H FOUR CHARACTER ASCII CONVERSION
Wandelt einen String,der eine 4-stellige Hexadezimalzahl im ASCII-Format darstellt in eine hexadezimale Zahl um und speichert diese im HL-Register.Das DE-Register muß die Anfangsadresse der hexadezimalen Zahl,die im ASCII-Format gegeben ist,enthalten.Diese Routine wandelt also eine 4-stellige Stringkette z.B. "3410") in eine hexadezimale 16-Bit Zahl um.Bei einem Fehler,der durch eine ungültige Stringkette hervorgerufen werden kann,wird das CARRY-Flag auf "1" gesetzt.

041FH TWO CHARACTER ASCII CONVERSION
Wandelt einen String,der eine 2-stellige Hexadezimalzahl im ASCII-Format darstellt,in eine hexadezimale Zahl um und speichert diese im Akkumulator.Das DE-Register muß die Anfangsadresse des zweistelligen String enthalten.Ansonsten siehe FOUR CHARACTER ASCII CONVERSION.

0436H WRITE HEADER
wie 0021H

0475H WRITE DATA
wie 0024H

04D8H READ HEADER
wie 0027H

04F8H READ DATA
wie 002AH

0577H BELL
wie 003EH

0588H VERIFY
wie 002EH

05FAH TWO CHARACTER PRINT
Ab der momentanen Cursorposition werden zwei Character,gegeben durch die ASCII-Werte des Inhalts vom HL-Register,auf dem Bildschirm ausgegeben.

069FH CASSETTE MOTOR ON
Schaltet den Cassettenrekordermotor ein.

0700H CASSETTE MOTOR OFF
Schaltet den Cassettenrekordermotor aus.

0759H 107 MICROSECOND DELAY
107 Microsekunden-Zeitverzögerung.

07A8H M-BEFEHL

07E6H GET LINE
wie 0003H

087CH CARRIAGE RETURN AND NEW LINE
Diese Routine führt ein Carriage-Return aus und positioniert den Cursor am Anfang der nächsten Zeile.

0893H PRINT MESSAGE
wie 0015H

08A1H PRINT MESSAGE
wie 0018H

08BDH GET KEY
wie 001BH

0918H NEW LINE
wie 0009H

0920H PRINT SPACE
wie 000CH

0924H PRINT 10 SPACES
wie 000FH

0935H PRINT CHARACTER
wie 0012H

0996H 7 MILLISECOND DELAY
7 Millisekunden-Zeitverzögerung.

09E0H NEW LINE
wie 0006H

0A32H SHIFT,BREAK,CONTROL
wie 001EH

0BB9H ASCII TO DISPLAY
Wandelt das im Akku gespeicherte ASCII-Zeichen in den Display-Code um und speichert diesen im Akku ab.

0BCEH DISPLAY TO ASCII
Wandelt das im Akku befindliche Display-Code Zeichen in den ASCII-Code um und speichert dieses im Akku ab.

0DA6H VERTICAL BLANKING CHECK
Warten auf die vertikale Austastlücke.Danach Rücksprung.

0DB5H CHARAKTER PRINT
An der momentanen Cursorposition wird das im Akku gespeicherte Display-Code Zeichen auf dem Bildschirm ausgegeben.

0DDCH DISPLAY CONTROL
Mit Hilfe dieser Routine kann man den Bildschirm vielfältig verändern.Dabei enthält der Akkumulator das entsprechende Kontrollzeichen.Die verschiedenen Kontrollzeichen und ihre Wirkung sind folgende:

Akku	Wirkung
C0H	Scrollen des Bildschirmes um eine Zeile nach unten.
C1H	Cursor wird eine Zeile nach unten bewegt.
C2H	Cursor wird eine Zeile nach oben bewegt.
C3H	Cursor wird um ein Zeichen nach rechts bewegt.
C4H	Cursor wird um ein Zeichen nach links bewegt.
C5H	Der Cursor wird an die HOME-Position (D000H) gesetzt.
C6H	Das Video-RAM und das Farb-RAM werden gelöscht,der Cursor wird an die HOME-Position (D000H) gesetzt.
C7H	Ein Character hinter dem Cursor wird gelöscht und der Cursor wird an diese Position gesetzt.Selbe Funktion wie die Taste 'DEL'.
C8H	Ein Character wird an der momentanen Cursorposition eingefügt und alle dahinter befindlichen Zeichen werden nach rechts verschoben.Selbe Funktion wie die Taste 'INST'.
C9H	Schaltet die Tastatur vom Graphik-Modus in den Alpha-Modus.
CDH	Ein CARRIAGE RETURN wird durchgeführt und der Cursor wird an den Anfang der nächsten Bildschirmzeile gesetzt.

0FB1H Die momentane Cursorposition wird in eine Video-RAM Adresse umgewandelt und im HL-Register abgespeichert.

0FCBH VERIFY BEFEHL

0FD8H Löscht den Speicher ab der in HL gegebenen Adresse.Die Länge des zu löschenden Bereiches gibt das B-Register an.Löschen bedeutet den Speicherbereich mit Data 00H laden.

Der Monitor benötigt den RAM-Bereich von 1000H-11FFH als Arbeitsbereich,d. h. in diesem Speicherbereich werden notwendige Daten abgelegt.

MONITOR HILFSZELLENBELEGUNG (ARBEITSBEREICH)

Adresse:	Bedeutung:
1000H-1037H	frei
1039H	LOW-Byte des Interrupt-Vektors.
103AH	HIGH-Byte des Interruptvektors. Bei einem Interrupt wird zu dem hier angegebenen Interruptvektor (Adresse) gesprungen.
103BH-10EFH	Stackbereich Normalerweise befindet sich bei jedem Programm in diesem Bereich der Stack.
10EFH	TOP OF STACK
10F0H-1107	Daten des Programmheaders
10F0H	FILECODE (Programmkennung) kennzeichnet Basicprogramm,Assemblerprogramm etc.
10F1H-1101	PROGRAMMNAME Der Programmname darf maximal 16 Zeichen lang sein und muß mit einem ODH (CR) abgeschlossen sein.Wenn in der Adresse 10F1H ein ODH steht,hat das Programm keinen Namen.
1102H	LOW-Byte PROGRAMMLAENGE
1103H	HIGH-Byte PROGRAMMLAENGE Die hier gespeicherte Programmlänge ist entscheidend für die Anzahl der Bytes,die bei der SAVE-Routine (0024H) auf Cassette abgespeichert werden.
1104H	LOW-Byte PROGRAMMANFANG
1105H	HIGH-Byte PROGRAMMANFANG Ab der hier gespeicherten Anfangsadresse wird begonnen das Programm bei Ansprung der SAVE-Routine (0024H) auf Cassette abzuspeichern.

1106H	LOW-Byte STARTADRESSE
1107H	HIGH-Byte STARTADRESSE Von dieser Adresse hängt ab,ob das eingeladene Programm automatisch ausgeführt wird.Wenn das Programm automatisch gestartet werden soll,muß in diesen beiden Adressen die richtige Startadresse des auszuführenden Programmes stehen.Die Startadresse muß größer oder gleich 1200H sein.Bei 0000H wird das Programm nicht gestartet.
1108H-116FH	frei
1170H	Merkzelle für Klein- oder Großbuchstaben. Inhalt 0 entspricht Großbuchstaben Inhalt 1 entspricht Kleinbuchstaben
1171H	CURSOR Y-POSITION (0-18H) Merkzelle für die Zeilenposition des Cursors.
1172H	CURSOR X-POSITION (0-27H) Merkzelle für die Spaltenposition des Cursors.
1173H-118BH	Je Bildschirmzeile ein Byte,das abgibt,ob diese Zeile der 2. Teil einer Doppelzeile ist.In diesem Fall ist der Inhalt der Merkzelle 1,sonst 0.
118EH	Merkzelle für die Ablage des Zeichens,auf dem der Cursor momentan steht.
118FH	Merkzelle für Cursorposition LOW-Byte
1190H	Merkzelle für Cursorposition HIGH-Byte nur beim Vorgängermodell MZ-80K benutzt.
1191H	Merkzelle ob Cursor oder Zeichen dargestellt ist. 0=Zeichen,1=Cursor nur beim Vorgängermodell MZ-80K benutzt.
1192H	Displaycode des Cursorzeichens
1193	STRING FLAG nur beim Vorgängermodell MZ-80K benutzt.

1194H	Merkzelle für Anzahl der Zeichen in der Eingabezeile
1195H	Cassettenvorspannlänge LOW-Byte
1196H	Cassettenvorspannlänge HIGH-Byte Merkzelle für die Länge des Vorspannes beim Cassetten-schreiben bzw. lesen.
1197H	CHECKSUM beim Einlesen LOW-Byte
1198H	CHECKSUM beim Einlesen HIGH-Byte In dieser Merkzelle wird die CHECKSUM-Summe(Prüfsumme) beim Einlesen von Cassette gespeichert.Die Prüfsumme ist die Anzahl der "1"-Bits in den geschriebenen Daten-bytes.Diese Summe dient dazu,festzustellen,ob ein Auf-zeichnungsfehler vorliegt.
1199H	CHECKSUM beim Schreiben LOW-Byte
119AH	CHECKSUM beim Schreiben HIGH-Byte Merkzelle für die CHECKSUM-Summe beim Schreiben auf Cassette.CHECKSUM siehe CHECKSUM beim Einlesen.
119BH	Merkzelle für AM/PM (Uhr) AM=0,PM=1
119CH	Merkzelle Uhr läuft Bei Inhalt F0H läuft die eigebaute Uhr..
119DH	Merkzelle KEY-SOUND Merkzelle für Ton erzeugen bei Tastendruck. Ton=0,kein Ton=FFH
119EH	Speicher für Musiktempo
119FH	Speicher für Tonlänge
11A0H	Speicher für Oktaven-Nummer
11A1H	Speicher Tonfrequenz LOW-Byte
11A2H	Speicher Tonfrequenz HIGH-Byte Der Inhalt dieser beiden Zellen ist entscheident für die Tonhöhe bei Ansprung der Monitor-Routine 0044H.
11A3H-11F3H	Eingabebuffer der Tastatur
11F4H-11FFH	frei

Was ist Maschinensprache ?

Dieses Kapitel soll dem Benutzer des MZ-700 helfen,die Geheimnise der Z-80 Maschinensprache zu lösen.Zuerst wird eine allgemeine Einführung in die Z-80 Mascchinensprache gebracht,welche dann systemspezifisch vertieft. wird.Dieses Kapitel ist so gestaltet,daß auch MZ-80K/A Besitzer es benutzen können.Als Vorraussetzung für alle drei Computertypen ist der Besitz einer Maschinensprache mit Disassembler.Sollte diese nicht vorhanden seien,so kann man jene für DM 90,- bei uns beziehen.

1) Das Dualssystem

Die Kenntnis des Dualsystems ist die Grundlage zur richtigen Maschinenprogrammierung.Wie Sie wahrscheinlich schon wissen,kann der Computer nur die beiden Zustände 0 und 1 verarbeiten.Deshalb ist es auch sinnvoll und nur konsequent,ein Rechensystem zu entwickeln ,das nur mit diesen beiden Ziffern operiert.Das Dualsystem ist genau so aufgebaut wie unser altes Dezimalsystem.Zur Erinnerung.Sämtliche Zahlen unseres Zahlensystems können als Folge von Potenzen dargestellt werden.Als Basis dient im Zehnersystem die Zahlen 10.Jede Zahl kann also als eine Summe der einzelnen Zehnerpotenzen dargestellt werden.Beispiel:174 = 1*100 + 7*10 + 4*1
noch weiter schematisiert wie folgt :174 = 1*10↑2 + 7*10↑1 + 4*10↑0

ALLGEMEIN:

ZAHL= X*10↑N + Y*10↑N-1 +....+ Z*10 + A*1

Dieses stellt nur eine mathematische Fassung dessen dar,was wir als selbstverständlich halten.Genau so ist es mit dem Dualsystem.Dort ist die Zahl 2 die Basis,entsprechend kann man nur mit zwei Ziffern rechnen.
Eine Dualzahl ist genau wie die Dezimalzahl aufgebaut.

ALLGEMEIN:

ZAHL= X*2↑N + Y*2↑N-1 +....+ Z*2 + A*1

Als Beispiel wollen wir die Dezimalzahl 27 als Dualzahl berechnen.
Zuerst müssen wir erst einmal die Zweierpotenzen kennen.

$2 \uparrow 0 = 1$
$2 \uparrow 1 = 2$
$2 \uparrow 2 = 4$
$2 \uparrow 3 = 8$
$2 \uparrow 4 = 16$
$2 \uparrow 5 = 32$

Die Zahl 27 wird als Dualzahl als eine Summe der Zweierpotenzen versucht darzustellen. $27 = 16 + 8 + 2 + 1$
$27 = 2 \uparrow 4 + 2 \uparrow 3 + 2 \uparrow 1 + 2 \uparrow 0$

Man verucht dieses mit einer Tabelle in den Griff zu bekommen.

128	64	32	16	8	4	2	1	Dez.zahl
0	0	0	1	1	0	1	1	27

Die Zahl 27 wird also als Dualzahl 11011 lauten.Als anderes Beispiel die Dezimalzahl 111.Wir schreiben uns also wieder unsere Tabelle auf,wo wir oben die Zweierpotenzen eintragen,und daunter unsere 0 bzw 1,also ob die jeweilige Zweierpotenz die Summe mitbildet.

128	64	32	16	8	4	2	1	Dez.zahl
0	1	1	0	1	1	1	1	111

Die Zahl 111 wird also als Dualzahl 1101111 lauten.

Hier nochmal eine einfache Methode um zu bestimmen ob man eine 0 oder eine 1 unter die Zweierpotenz setzen muß.Wir wollen dieses am Beispiel der Dezimalzahl 111 nachvollziehen.Zuerst suchen wir in der Tabelle die Zweier potenz,welche niedriger ist als die Dezimalzahl (64) .Dort setzen wir eine 1 ein.Dann ziehen wir diese Zweierpotenz von der Dezimalzahl ab. 111-64 = 47.Jetzt machen wir das gleiche nochmal,nur ist die Dezimalzahl jetzt 47.Die nächstniedrigere Zweierpotenz ist 32.Dort setzen wir auch eine 1 ein und subtrahieren 47-32 = 15.wir setzen bei 8 eine eins ein und ziehen 8 von 15 ab.Dieses Verfahren wird bis zur letzten Stelle durchgezogen.
So erhält man jede Dualzahl,welche man haben will und somit ein Rechensystem für den Computer zur internen Verarbeitung.

2) Das Hexadezimalsystem

Sicher ist es für den Computer eine feine Sache , wenn er gleich ein Rechensystem zur Verfügung hat,was er gut verarbeiten kann,jedoch geben sich dem Benutzer des Systems einige Probleme auf.Welcher Mensch kann schon aus einem Programm,wo hunderte Nullen und Einsen auftauchen schlau werden.Deshalb hat der Z-80 eine anderes Rechensystem,nämlich das Hexadezimalsystem,welches die Zahl 16 zur Basis hat.Jetzt werden Sie sich natürlich fragen,warum gerade die Zahl 16 ?.Ganz einfach,weil 16 eine Zweierpotenz von 2 ist und deshalb die Umrechnung dieser beiden Zahlensysteme keinerlei Schwierigkeit bietet,und weil das Hexadezimalsystem unserem Zehnersystem noch am nächsten kommt.Eine weitere Frage müßte sich nun stellen.Wie werden die Zahlen 10 bis 15 dargestellt ?.Hier hat man nun die Buchstaben A-F genommen.10 ist also A,B ist 11, C ist 12 usw.
Die Berechnung einer Hexzahl erfolgt genau wie die einer Dualzahl,nun müssen wir diesmal die Sechzehnerpotenzen nehmen.Unsere Potenzen schreiben wir wieder in eine Tabelle.

4096	256	16	1	Dez.zahl

Als Beispiel nehmen wir die Dezimalzahl 83,welche wir als Hexzahl berechnet haben wollen.

Zuerst machen wir uns eine Tabelle mit den Sechzehnerpotenzen.

4096	256	16	1	Dez.zahl
0	0	5	3	83

Wir halten uns wieder an das Rechenschema und suchen die nächstkleinere Sechzehnerpotenz.wir erhalten 16.5*16 ergibt 80.wir ziehen diese 80 ab. und es bleiben 3 übrig.Wir erhlten als Hexzahl 53.Wir machen diesmal eine überprüfung nach dem allgemeinen Zahlensystemsatz,welchen wir nun für das Hexadezimalsystem anwenden. ZAHL= X*16↑N +Y*16↑N-1 +...+ A*16 +B
5*16+3 = 83 .So können wir jederzeit überprüfen,ob das erhalten Ergebnis stimmt.Nehmen wir als nächste Beispiel eine größere Dezimalzahl,z.B. 15311
Zuerst suchen wir die nächstkleinere Potenz,also 4096,welche dreimal hineinpaßt.Dann ziehen wir das Produkt von 4096*3 von 15311 ab.
15311-12288 = 3023
Wir machen dann wie gewohnt weiter und untersuchen mit der nächsten Potenz,nämlich 256,welche elfmal in 3023 hineinpaßt.Wir subtrahieren dann wieder wie gewohnt.3023 -(11*256) = 207
Dann teilen wir 207 Durch 16 (ergibt 13) und der Rest ist 15
Wir haben dann folgende Potenzzerlegung erreicht

15311 = 3 *16↑3 + 11*16↑2 +12 *16↑1 +15*1

Daraus ergibt sich die Hexzahl 3BCF.Die einzelnen Multiplikanden werden dann nätürlich in Hexziffern umgeformt und zur Hexzahl zusammengezogen

Beim Z-80 ist als höchste Adresse im Speicher die Hexzahl FFFF vorgesehen. 15*4096 + 15 *256 +15*16 +15 = 65535.Der freie Speicher beträgt demnach 65536 Speicherzellen,womit wir beim nächsten Thema ankommen, nämlich den Bytes.

3) Bit und Byte

Wie schon gesagt,wird eine Speicherstelle im Computer Byte genannt.Innerhalb eines Bytes kann eine achtstellige Dualzahl gespeichert werden.
Ein Byte besteht also aus acht Bits.Jedes Bit kann eine 0 oder 1 speichern
Durch die achtstellige Dualzahl kann jedes Byte 2↑8 = 256 verschiedene Informationen speichern.
Die Bits sind die kleinste logische Einheit,die der Computer kennt.
Speicherinhalte werden für gewöhnlich als Hexzahlen ausgegeben.

4) Der Prozessor Z-80

Herzstück eines jeden Microcomputers ist der Prozessor,welche bei uns der Z-80 Prozessor ist.Der Z-80 Prozessor ist ein acht-Bit Prozessor,ebenso wie sein harter Konkurrent,der 6502 (welcher in APPLE,ATARI ,CBM u.a zu finden ist).Jeder Prozessor hat seinen eigenen Befehlssatz,mit dessen Hilfe Maschinenprogramme abgearbeitet werden.Der Befehlsvorrat des Z-80 umfaßt etwa 400 Befehle,welche aus einer Folge von 1 bis 4 Bytes bestehen
Es gibt noch ein paar Zusatzbefehle,auf welche später nochmal eingegangen wird.

4.1) Die Register

Register sind eine besondere Einrichtung im Prozessor.Mit Register werden Werte vom bzw. zum Speicher transportiert,es wird mit Registern gerechnet, verknüpft und v.a. Es gibt ein - sowie zweibyteregister,was den Z-80 z.B. dem 6502 vorraus hat.Das wichtigste Register ist der Akkumulator,das A-Register.Man kann sämtliche arithmetisch-logische Operation mit dem Akku und einem anderem Partner vollziehen.Als weitere Register sind B,C,D,E,H,L zu nennen,welches ebenfalls Ein-Byteregister sind.Man kann mit diesen Register allerdings nicht so hervoragend operieren wie mit dem Akku.Des weiteren gibt es die Regiter IX und IX.(Zweibyteregister), das Flagregister F,den Stackpointer SP,den Programmcounter PC und zur Bedienung der Hardware die Register R (Refreshregister) und I (Interruptregister).Als Besonderheit sind die Doppelregister von A,B,C,D,E,H,L,F die ' Register hervorzuheben.

4.1.1) Der Akkumulator

Wie schon angedeutet nimmt der Akkumulator eine besondere Stellung im Registersatz ein.Der Akku ist ein 8 Bit Register,welcher sich von den anderen 8 Bit Registern durch eine größere Auswahl an Befehlen unterscheidet.So werden sämtliche 8-Bit arithmetisch-logische Befehle über den Akku abgewickelt.Dabei wird der Akku immer mit einem anderen Partner, entweder ein anderes Register oder der Speicher verknüpft.
Auch ist der Akku als einziges Register in der Lage,Werte von bzw. zum Speicher zu transportieren.Dieses macht ihn zu einem wichtigen Bestandteil im Prozessor.

4.1.2) Weitere 8-bit Register

Außer dem Akku hat der Prozessor noch weitere 8-bit Register zu Verfügung.Nämlich: B,C,D,E,H,L
Die Funktionsweise dieser Register ist dem Akku wohl ähnlich,doch ist der Befehlsatz erheblich eingeschränkt.So bilden diese 8-bit Register nicht die Grundlage bei arithmetisch-logischen Operationen,sondern immer nur den Verknüpfungspartner des Akkus.

4.1.3) Die Flags (F-Register)

Das F-Register ist in 8 Bits aufgeteilt,den Flags.

7	6	5	4	3	2	1	0
S	Z	-	H	-	P	N	C

Davon werden nur 6 Flags benutzt.(Siehe obere Skizze)
Die Namen der Flags

S Sign Flag
Z Zero Flag
H Halfcarry Flag
P Parity Flag
N Subtraktion Flag
C Carry Flag

Grundsätzlich werden Flags benutzt um die Ergebnisse irgenswelche Operationen,wie z.B. Addition oder ähnliches zu verdeutlichen.Als Beispiel haben wir im Akku 20(Hex),ebenfalls im B-Register auch 20(H).Wenn wir also die Operation Subtrahiere B von A durchführen , so wird natürlich 0 im Akku stehen.Wenn nach einer solchen Operation 0 im Akku steht,wird das Zero-Bit mit 1 gesetzt,anderenfalls zurückgesetzt.

4.1.3.1) Das Carryflag

Das Carryflag ist das Flag,welches den übertrag bei einer logischen Operation anzeigt,daß heißt,man kann es sich als fiktiven neuntes Bit vorstellen.Als Beispiel wollen wir den Akku mit dem C-Register addieren.A enthalte 90 (Hex) und C entahlt 83 (hex).Die Addition dieser beiden Werte würden 113(Hex) ergeben,aber der Akku speichert ja nur 8-Bit. Im Akku steht dann 13,und die eins steht im Carry Flag.

4.1.3.2) Das Subtraktions Flag

Dieses Flag wird vom Programmierer selbst nicht genutzt,sondern nur bei speziellen dezimalarithmetischen Befehlen als internes Vorzeichenbit genutzt,und ist daher für die Programmierung irrelevant.

4.1.3.3) Das Halfcarry Flag

Das H Flag zeigt den möglichen übertrag vom vierten auf fünfte Bit an.Die Bedeutung dieses Flags ist sehr gering und für den Maschinenprogrammierer gibt es auch keinen direkte Möglichkeit dieses Flags zu kontrollieren.

4.1.3.4) Parity Flag

Das Parity oder auch überlauf Flag hat zwei Funktionen.

1) Die Parität des Ergebnisses.Parität heißt soviel wie Quersumme. Das heißt,man zählt die Einsen innerhalb eines Bytes.Ergibt die Summe der Einsen eine gerade Anzahl,so wird das P-Flag gesetzt,andernfalls wird es zurückgesetzt.

2) Die zweite wesentliche Funktion ist die des überlaufes.Es zeigt an,ob einer Addition oder Subtraktion das Vorzeichenbit "fälschlicherweise" verändert wurde,da das Ergebnis in das Vorzeichenbit überlief.

4.1.3.5) Das Zeroflag

Das Zeroflag ist eines der am häufigsten benutzten Flags,welches nach einer logischen Operation anzeigt,ob das Ergebnis null ist. (siehe obige Erwähnung)

4.1.3.6) Das Signflag

Dieses Flag wird auch als das Vorzeichenbit betrachtet.Das heißt,daß ein in sieben Datenbits und ein Vorzeichenbit geteilt wird.Ist das Signflag gesetzt,so hat man ein negatives Ergebnis,anders herum natürlich ein positives.

4.1.4) Das R-Register

Das R-Register,oder auch Refresh-Register ist ein vom Programmierer nicht benutztes Register,welches ein Zähler darstellt,um Speichern nach einer bestimmeten Zeit einen Refreshimpuls zu geben.Das heißt,da dieses Register konstant von 7F Hex auf 0 heruntergezählt wird,um dann einen Impuls zu geben.Man kann dieses Register gut als Zufallsgenerator verwenden,wenn man auf es zugreift.

4.1.4) Das I-Register

Das I-Register,oder auch Interrupt-Register ist von unwesentlicher Bedeutung für den Programmierer,es zeigt nur einen gewissen Status in der Betriebsystemebene an.

4.1.5) Das PC-Register

Der sogenannte Programmcounter ist ein Zähler,welcher anzeigt,wo im Speicher der Prozessor gerade arbeitet.Zum Programmcounter hat man keinen direkten Zugriff.Für den Programmierer bleibt es ein mehr oder weinger fiktives Register (16 Bit).

4.1.6) Das SP-Register

Der sogenannte Stackpointer ist ein Sechzehnbitregister,welcher auf einen Adresse im Speicher adressiert ist.Dort befindet sich der Stack, ein durch eben den Stackpointer definierter Workbereich,in den Daten gebracht,und von dem Daten geholt werden.Für die Verwaltung des Stacks gibt es spezielle Befehle,auf die später noch einmal eingegangen wird.

4.1.7) Das HL-Register

Das HL-Register ist ein Sechzehnbitregister welches sich aus den zwei Achtbitreigistern L und H zusammensetzt.Dabei bildet zweckmäßigerweise das L-Register das Low-Byte und H das High Byte.Zur Auffrischung.Low ist niederwertig und H hochwertig.Hier kommen wir auf etwas zu sprechen,was bei jeder Sechzehnbitnotation wichtig ist.Im Speicher wird zuerst das Low,und dann das High Byte geschrieben.Wenn also H 06 Hex und L F3 enthalte,so hat das Register HL den Wert 06F3.Wird dieses aber im Speicher geschrieben,so wird zuerst F3 und dann 06 schrieben.Zurück zum HL Register Dieses hat ähnlich dem Akku bei den Achtbitregistern eine prädestinierte Stellung unter den Sechzehnbitregistern.So werden fast alle arithmetisch-logischen Befehle über HL abgewickelt.Deshalb ist das HL-Register ein unentbehrliches Sechzehnbitwerkzeug.

4.1.8) Das DE und BC Register

Diese beiden Register sind wie HL Sechzehnbitregister und bestehen aus den Achtbitregistern D und E,bzw B und C.In der Art der Adressierung sind sie mit HL identisch,doch ist der Befehlskomfort dieser beiden Register doch erheblich geringer.Was noch bei allen Sechzehnbitregistern zu erwähnen wäre,ist das wenn man z.B. das Register B ändert,sich auch automatisch der Wert des BC Regiters ändert.Ein Flüchtigkeitsfehler,welcher bei der Programmierung in Maschinensprache gerne gemacht wird.

4.1.9) Das IX und IX Register

Ebenfalls wie HL Sechzehnbitregister.Was hier zu beachten ist ist die Art der Nomenklatur,welche ich persöhnlich für etwas schlecht gewählt halte,da man den I-Teil von z.B. IX als mit Interruptregister verwechseln könnte.Deshalb wird das L-Byte von IX auch LX und das High Byte HX genannt.Ebenso verfährt man bei IY,wo dann LY und LX entstehen.Wichtig ist das dieses Achtbitregister sind.Man nennt IX und IX auch Indexregister.Die Indexregister sind in Ihrem Befehlsatz HL sehr ähnlich,doch besteht ein Indexbefehl aus mindestens drei Byte,wogegen bei HL häufig ein Byte ausreicht.Deshalb benutzt man die Indexregister nicht so häufig da dadurch ein Programm sehr lang wird.über die Art der Indexadressierung reden wir in einem anderen Teil des Kapitels,da dieses recht kompliziert ist und dem Maschinensprachbeginner wohl einiges Kopfzerbrechen bereitet

4.1.10) Die Doppelregister

Eine Besondertheit des Z-80 sind die sogenannten Doppelregister.Es sind Duplikate der Register A,B,C,D,E,H,L,F vorhanden,Welche normalerweise nicht benutzt werden.Durch einen Tauschbefehl kann man die Doppelregister mit den normalen Registern umtauschen,so das der Prozessor mit den Werten der Dcppelregister weiterrechnet.Wenn man den Tauschbefehl zweimal anwendet,wird logischerweise der Orginalzustand wieder hergestellt.

4.1.11) Registerübersicht

Achtbitregister:	B C D E H L A F
Sechzehnbitregister:	BC DE HL
Indexregister:	IX IY
Doppelregister:	B' C' D' E' H' L' A' F'
Hardwareregister:	R I
Restregister:	PC SP

4.2) Mnemonic und Opcode

Ein weiteres wichtiges Kapitel in der Prosezzorarchitektur ist der Zusammenhang zwischen Symbol und Bedeutung eines Maschinensprachebefehles In der Maschinensprache gibt es ganau wie in Basic Befehle,welche eine Bedeutung haben.In Basic hat der Befehl DIM z.B. die Bedeutung ein Array zu öffnen.In Maschinensprache hat jeder Befehl einen Hexcode,besteht also aus einer Folge von hexadezimalen Zahlen.Bei der Ausführung werden die hexadezimalen Zahlen vom Prozessor verarbeitet und als Befehle interpretiert.Ein Beispiel.Der Hexcode 00 hat die Bedeutung NOP.(No OPeration) Trifft der Prozessor in einem ML Programm auf 00 so wird er dieses als ein NOP auffappen un nichts tun.Trifft der Prozessor z.B. auf ein 80 Hex, so wird er dieses als ADD A,B auffassen,also addiere B zu A,womit wir gleich zum mnemotechnischen Code kommen.Der Mnemonic eines Befehles ist der für den Programmierer verständliche Code,also in den obigen Beispielen NOP und ADD A,B.Der für den Computer relevanten Teil nennt man Opcode, also im Beispiel 00 bzw. 80.Als Programmierer ist an und für sich nur der Mnemonic verständlich.Deshalb gibt es Programmierwerkzeuge,welche dem Programmier helfen nur mit diesem Code zu arbeiten.Einen Umsetzer von Mnemonic in Hexcode nennt man Assembler,den umgekehrten Vorgang beherrscht der sogenannte Disassembler.Als Programmierer kann man normalerweise den symbolischen Code im Assembler eingeben,welcher dann automatisch in den lauffähigen Z-80 Code umgewandelt wird.Wenn man allerdings ein fertiges Maschinenprogramm hat,wird jenes vom Disassembler in ein lesbares Programm verwandelt.Jede gute Maschinensprache hat daher einen Disassembler.Wir werden in diesem Kapitel einen Maschinensprache mit Disassembler verwenden,da die Assemblerprogrammierung etwas schwieriger für den Anfänger ist.Da wir nur kleine Programme schreiben werden,reicht die Maschinensprache vollkommen aus.Machinensprache wird fachterminologisch gerne als ML bezeichnet,welche wir ab jetzt auch tun werden.

4.3) Speicherstruktur

Besonders wichtig ist natürlich die Adresse,wo das Programm steht,womit wir auf das Thema Speicher zu sprechen kommen.Der MZ-700 hat in der Normalbetriebsart 64 KByte zu Verfügung.Der Programmierbereich beginnt ab 1200 (Hex).Dort sollen auch unsere Programme alle beginnen.Zu jeder Adresse gehört natürlich auch ein Inhalt,welcher entweder ein Programm, oder auch ein Datenbereich sein kann.Wenn der Computer angeschaltet ist, und aufwärts nichts.(Bei MZ-80K kann 00 sich mit FF abwechseln).
Wir wollen dieses Disassemblieren.Wir geben D ein.Anschließend wir die Anfangsadresse ein.Je nach ML-Version wird jetzt noch eine Endadresse abgefragt.Sämtliche Eingaben sind natürlich hexadezimal.Eine richtige Eingabe sieht wie folgt aus   *D from:1200 to XXXX
Die Endadresse XXXX ist natürlich beliebig wählbar,z.B.1204 (Hex)
Nun wird folendes Bild ausgegeben

```
1200      00      NOP
1201      00      NOP
1202      00      NOP
1203      00      NOP

↑         ↑       ↑
Adresse   Opcode  Mnemonic
```

4.4) Adressierungsarten.

In den Adressierungsarten hat der Z-80 nicht so viele Möglichkeiten,wie sein Gegenüber,der 6502.doch gibt es allgemeines und spezielles hierüber zu sagen,was von nicht unerheblicher Relevanz ist.Zuerst etwas allgemeines über die generelle Adressierung.Der Z-80 hat innerhalb der Mnemonic immer drei Teile.

A) Art des Befehles
B) Zieladresse/Register
C) Quelladresse/Register

Wir wollen dieses an einem Beispiel erläutern.
Beispiel: LD B,A

Dieses ist ein Ladebefehl,wo der Imhalt von A nach B tranferiert wird.In Basic ungefähr:B=A.Hieran sieht man,wie ein solcher Befehl aufgebaut ist. Zuerst kommt die Art des Befehles,also in diesem Fall ein Ladebefehl.Dann folgt Ziel des Wertes,Also das B-Register,anschließend Quelle,Also Akku. Dieses ist eine Registeradressierung,was jedem wohl noch einleuten mag. Anders ist es schon bei der Speicheradressierung.Ein Beispiel

Beispiel: LD A,(HL)

In diesem Falle ist wieder ein Ladebefehl vorhanden.Rein schematich gesehen stellt der Akku schon mal das Ziel des Achtbitwertes dar.Nur die Quelle bedarf noch einiger Interpretation.Wir sehen das die Quelle (HL) lautet.Generell gesagt wird mit der Klammer gezeigt,das eine Speicheroperation vollzogen wird.Speicheradressierung sind grundsätzlich 16 Bit-Adressierungen.In diesem Falle werden die 16 Bit durch den Wert im Register HL dargestellt.Das heißt,das der Wert in HL als Speicheradresse interpretiert wird,und aus dieser durch HL bestimmten Adresse wird ein Achtbitwert in HL geladen.Nun noch ein letztes Beispiel.

Beispiel: LD (1000),HL

Wiederum ist hier ein Ladebefehl vorhanden.Schauen wir uns nun erstmal das Ziel an.An der Klammer sehen wir,das der Speicher gemeint ist.In diesem Falle die Adresse 1000 (Hex).Das Quellregister ist in diesem Falle HL.Mit einem Wort gesagt,wird der 16 Bitwert von HL in die Speicheradressen 1000 und 1001 geladen.(HL hat ja 2 Byte=16 Bit).Doch Vorsicht ! Bedenke die Sechszehnbitnotation im speicher,zuerst kommt L-Byte,dann H-Byte.Enthalte HL den Wert 2400 (Hex) so steht nach Vollstreckung der obigen Operation in 1000 der Wert 00,in 1001 ist 24 (alles Hex)

Als letzies wollen wir auf die schon angedeuteten Adressierungen der Indexregister IX und IY eingehen.IX und IY kann man als großen Bruder von HL auffassen.So fügt man vor den normalen Opcode bei IX ein DD (Hex) und bei IY ein FD.Beispiel INC HL = 23 (Hex)

```
INC IX  = DD 23 (Hex)
```

Dieses gilt nur für Z-80 Befehle,wo die Dreiteilung des Mnemonic nicht gewährleistet ist.(Siehe obiges Kapitel).Ansonsten tritt hier das Phänomem der indirekten Adressierung auf.Außer dem DD bwz. FD vor dem normalen Opcode,wird hinter den normalen Opcode ein weiteres Byte gesetzt, welche die indirekte Adressierung deklariert.Beispiel:

Beispiel:FD 7E 01 LD A,(IX+01)

Dieses hat äußerlich sehr viel Ähnlichkeit mit dem vorhin dokumentierten LD A,(HL).Unterschiedlich ist hier nur das IX Register und das mysteriöse +01 hinter dem IX.Zuerst zur Berechnung des +01 hinter dem IX

4.4.1 Berechnung der Indexadressierung

Die Berechnung des Achtbitwertes ist sehr einfach.Man nimmt das Byte,und sieht nach,ob das siebte Bit 1 ist,wenn ja ist die Zahl negativ.Dadurch ist allerdings nur ein Rechenbereich von +127 bis -128 möglich.Im oberen Falle wird 01 zum IX Register addiert,anschließend wird aus der gebildete Sechszehnbitwert als Speicheradresse aufgefasst und der Inhalt jenes in den Akku geladen.Als anderes Beispiel soll +F0 hinter dem IX stehen,und IX habe den Wert 2000.Zuerst wird der Indexwert berechnet. Das siebte Bit ist gesetzt.Man negiert diesen Wert (Komplement) und zieht diesen Wert (10 hex) von 2000 (Hex)ab,um somit den Gesamtwert von 1FF0 zu erhalten.Aus der Adresse 1FF0 wird dann der Inhalt in den Akku geladen.

Beispiele :	IX Wert	Indexwert	gebideltete Adresse.	
	3000	10	3010	(alles Hex)
	3000	00	3000	(alles Hex)
	3000	C0	2FC0	(alles Hex)

Doch der noch unerfahrene Programmierer sollte zuerst sich mit dem Register HL üben,da die Indexadressierung über IX nicht leicht ist.

4.5) Befehle des Z-80

In Anhang wird der Beutzer eine Liste aller Z-80 Befehle finden.Der Benutzer mag dieses als Nachschlagewerk nutzen.Die Befehle sind nach dem Mnemoniccode alphabetisch geordnet.

4.6) Binärarithmetik

Die binäre Logik ist innerhalb der Machinensprache eine wichtige Sache.. Meistens dreht es sich um Verknüpfungen von Bytes,womit man bestimmte Zustände des Akkus besser analysieren kann.Es gibt folgende Verknüpfungsbefehle.

1) AND
2) OR
3) XOR

Zuerst etwas zur Grundlage von Verknüpfungen.Es werden immer zwei Bits nach einem bestimmten Schema verknüpft,worauf dann das Bit ein je nach Operation abhängigen neuen Wert erlangt.Dabei geht man nach Tabellen vor. Als Beispiel die Tabelle von OR

Bit 1	Bit 2	Endbit
0	0	0
1	0	1
0	1	1
1	1	1

Zur Erläuterung.Wenn also Bit 1 den Wert 1 hat und Bit 2 den Wert 0,dann erhält das Endbit den Wert 1.In Worte gefasst kann man auch sagen,das Endbit erhält den Wert 1,wenn mindestens eines der beiden Ursprungsbits 1 ist.In der Maschinensprache verknüpfen wir immer ein Byte mit dem anderen.Als Beispiel nun OR A,B.Der Akku enthalte den Wert 45 (hex) und B den Wert 08 (Hex).Wir schreiben nun diese beiden Werte in Binärform untereinander.

```
Register A :01000101     = 45(Hex)
Register B :00001001     =  9(Hex)
           ----------
Ergebnis   :01001101     = 4D(Hex)
```

Wir haben die untereinanderstehenden Bits nach der OR-Tabelle verknüpft und erhalten im Akku als neuen Wert 4D(Hex)
Hier nun die Tabelle für den AND Befehl.

Bit 1	Bit 2	Endbit
0	0	0
1	0	0
0	1	0
1	1	1

Wir nehmen nun die obigen Daten un verknüpfen sie nach der AND-Tabelle

```
Register A :01000101     = 45(Hex)
Register B :00001001     =  9(Hex)
            ----------
Ergebnis   :00000001     = 01(Hex)
```

Nach Vollendung des Befehle AND A,B steht im Akku also der Wert 01 (Hex)
Man kann den AND Befehl auch so formulieren.Nur wenn beide Ursprungsbits 1 sind erhält das Endbit 1.
Als letztes nun die Tabelle des XOR-Befehles.

Bit 1	Bit 2	Endbit
0	0	0
1	0	1
0	1	1
1	1	0

Wir nehmen nun die obigen Daten un verknüpfen sie nach der XOR-Tabelle

```
Register A :01000101     = 45(Hex)
Register B :00001001     =  9(Hex)
            ----------
Ergebnis   :01001100     = 4C(Hex)
```

Nach Vollendung des Befehles XOR A,B steht im Akku also der Wert 4C (Hex)
Man kann den XOR Befehl auch so formulieren.Nur wenn beide Ursprungsbits unterschiedlich sind,enthalt das Endbit 1.Ein beliebtes Anwendungsbeipiel für den XOR Befehl ist XOR A,A.Man 'xort' den Akku mit sich selbst. beide Verknüpfungdpartner sind in diesem Falle identisch.Da aber das Endbit nur bei unterschiedlichen Ursprungsbits 1 erhält,wird der Endwert hier immer 0 werden.

4.7) logische Operationen.

Es gibt einige Operationen ,welche dem Programmierer bei Rechnungen unterstützen.

4.7.1) Der ADD Befehl

Der ADD Befehl ist ein einfacher Additionsbefehl.Z.B. ADD HL,BC addiert BC zu HL,in welchem dann das Ergebnis steht.(16 Bit Operation !!!!)

4.7.2) Der ADC Befehl

Der ADC Befehl ist ein wie der ADD Befehl,doh wird zum Ergebnis noch der Inhalt des Carry Flags zugezählt.Setzt man das Carry Flag zurück,entspricht der ADC Befehl dem ADD Befehl.

4.7.3) Der SUB Befehl

Der SUB Befehl ist ein einfacher Subtraktionsbefehl.Z.B. SUB A,B subtrahiert den Wert in B vom Akku

4.7.4) Der ADC Befehl

Der SBC Befehl ist ein wie der SUB Befehl,dooh wird vom Ergebnis noch der Inhalt des Carry Flags abgezählt.Setzt man das Carry Flag zurück,entspricht der ADC Befehl dem ADD Befehl.

4.7.5) Der INC Befehl

Der INC Befehl zählt einfach das jeweilige Register um einen hoch,z.B. INC HL zählt das HL-Register um einen hoch.Wie in Basic X=X+1

4.7.6) Der DEC Befehl

Der DEC Befehl zählt einfach das jeweilige Register um einen runter,z.B. DEC HL zählt das HL-Register um einen runter.Wie in Basic X=X-1

4.7.6) Der CP Befehl

Der CP Befehl ist ein Vergleich eines Registers mit dem Akku.So wird verglichen ob der Akkuwert gleich ,kleiner oder größer ist.Beispiel Akku enthalte 99 (Hex) ,B enthalte ebenfalls 99,so ist nach Ausführung des Befehles CP A,B das Zeroflag gesetzt.Es wird im Geiste das B Register vom Akku abgezogen,und nach dem fiktiven Wert werden die Flags gesetzt.

4.8) Einzelbitverarbeitung

Es gibt im Z-80 Befehle,welche es ermöglichen,einzelne Bits abzufragen, zu setzen,bzw. zurückzusetzen.Zuerst wird die Art des Befehles ,dann das gewünschte Bit,und zuguterletzt das Quellregister angegeben.

4.8.1) Der SET Befehl

Der SET-Befehl ermöglicht es einzelne Bits zu setzen.Beispiel
Beispiel:SET 5,B
Hier wird das 5te Bit im B-Register gesetzt.Enthielt B vorher 0,so enthält es nun 20 (Hex).

4.8.2) Der RES Befehl

Der RES-Befehl ermöglicht es einzelne Bits zu zurückzusetzen.Beispiel
Beispiel:RES 5,B
Hier wird das 5te Bit im B-Register zurückgesetzt.Enthielt B vorher 23,so enthält es nun 03 (Hex).Enthielt der zurückzusetzende Bit vorher schon eine 0,so ändert sich nichts.

4.8.3) Der BIT Befehl

Der BIT-Befehl ermöglicht es einzelne Bits abzufragen.
Beispiel:BIT 0,A
Hier wird das 0te Bit im Akku auf seinen Inhalt abgefragt.Das Komplement des Inhaltes wird dabei in das Zeroflag gesetzt.Enthielt also das 0te Bit des Akku eine 0,so steht eine 1 im Zeroflage,andersherum würde ein 0 in das Z-Flag gestzt werden.

4.8.4) Rotationsbefehle

Rotationsbefehle sind Befehle,mit welchen man Register rotieren lassen, kann,das heißt ,z.B. bei linksrotieren wird Bit 0 nach Bit 1 geschoben 5 nach Bit 6,Bit 4 nach Bit 5 usw.Beim rechtsrotieren naturlich genau andersrum.über die einzelnen Möglichkeiten soll hier nicht weiter eingegangen werden,da sich die Rotationsbefehle nur unwesentlich voreinander unterscheiden.Man kommt meistens mit den Befehlen RLCA und RRCA aus (Akku links und rechts herum schieben).

4.8.5) Blockbefehle

Der Z-80 Prozessor hat die Besonderheit,Blockbefehle zu besitzen.Diese teilen sich in zwei Arten auf.

A) Blocktransferbefehle
B) Blocksuchbefehle

4.8.5.1) Blocktranfer

Der Blocktransfer soll am Beispiel des LDIR Befehles dargestellt werden. Es soll hierbei in Block im Speicher in einen anderen Bereich geschoben werden.Dabei ist die Anwendung des LDIR Befehles nur sinnvoll.Wir wollen den Bereich von 2000 (hex) bis 3000 (hex) nach C000 (hex verschieben. Für den LDIR Befehl sind dabei drei Parameter wichtig.

a) die Länge des Blockes Steht in Register BC
b) die Anfangsadresse des Ursprungsblockes Steht in Register HL
C) die Anafngsadresse des Endblockes Steht in Register DE

Sind alle Parameter gesetzt,kann man den LDIR Befehl anwenden.Im obigen Beispiel müßte folgendes Programm den Blocktransfer bewirken
Beispiel:

```
11 00 C0        LD   DE,C000
21 00 20        LD   HL,2000
01 00 10        LD   BC,1000
ED B0           LDIR
```

4.8.5.2) Blocksearch

Der Blocksearch soll am Beispiel des CPIR Befehles dargestellt werden. Dabei wird ein Zeichen,welches im Akku steht,innerhalb eines bestimmten Bereiches gesucht.Die Anfangsadresse steht in HL,Länge des Blocken in BC Als Beispiel soll das Zeichen 66 im Bereich von 3000 bis 4000 (hex) gesucht werden,wird es gefunden so wird an der Stelle abgebrochen und das Z-Flag ist 1,ansonsten bis zum Ende un das Zeroflag ist 0.
Beispiel:

```
LD   A,66
LD   HL,3000
LD   BC,1000
CPIR
```

4.8.6) Sprungbefehle

Grundsätzlich gibt es zwei Arten von Sprungbefehlen.Sprünge setzen ein Programm an anderer Stelle im Speicher fort,wie in Basic GOTO.Dabei gibt es noch Sprünge,welche von Bedingungen abhängig sind,also wenn das Z-Flag null ist ,dann springe dorthin.Dieses wird häufig nach logischen Operationen angewandt,z.B. um festzustellen,ob eine Schleife zu Ende ist.

4.8.6.1) Absolute Sprungbefehle

Absolute Sprungbefehle sind die einfacheren Sprungbefehle,z.B.
JP   3000 .Hier springt das Programm nach 3000 (hex). oder
JP   NZ,3000 .Hier springt das Programm nach 3000,wenn das Z-Flag nicht gesetzt ist.NZ bedeutet nicht Zero.N bedeutet immer Nicht.NC = Not Carry.

4.8.6.1) relative Sprungbefehle

Relative Sprungbefehle sind etwas komplizierter als absolute.Die Bedingundabhängigkeit ist gleich geblieben,doch ist die Adressierungsart anders. Sie ähnelt der in Kapitel 4.4.1 besprochenen indirekten Berechnung.Wir erinnern uns.Nach dem IX folgte ein Bytes,dessen Darstellung einen Zahlenbereich zwischen -128 und 127 erlaubte.Genau dieses ist beim relativen Sprung der Fall.Erst folgt das normale Byte für die Art des Jumps, und dann ein Byte für die indirekte Addressierung.Z.B.

```
38 06        JR  C,+06
```

In diesem Beispiel wird das Programm sechs Bytes weiter fortgesetzt,wenn das Carry Flag gesetzt ist.Der Vorteil von relativen Sprungen ist,dass das einmal so geschriebene Programm in jedem Speicherbereich läuft.Der Nachteil ist der geringe Erfassungsbereich von nur 256 Byte,wogegen der absolute Sprung den ganzen Speicher erfasst.
Eine Sonderform ist der Registerabhängige Sprung.Als Beispiel

```
JP   (HL)
```

Dieses bedeutet,das das Programm an der Stelle im Speicher fortgesetzt wird,welche momentan in HL steht.Hat HL den Wert 4000 (hex), so springt das Programm nach Ausführung des JP (HL) befhles nach 4000.Dieses gilt übrigens auch für die Register IX und IY.

4.8.6) CALL Befehle

Der CALL Befehl ist dem GOSUB Befehl in Basic identisch.Es wird in eine durch RET (Return) abgeschlossene Unterroutine gesprungen.Es gibt beim CALL ebenfalls Bedingungsabhängige Aufrufe wie z.B. CALL Z,xxxx.Also es wird die Unterroutine xxxx nur aufgerufen,wenn das Z-Flag gleich 1 ist. Das gleiche gilt für das RET.

4.8.7) Der DJNZ Befehle

Ein besonderer Sprungbefehl ist der DJNZ Befehl,welcher eine Art Schleifenbefehl darstellt.Hierbei wird das B-Register als Schleifenzähler benutzt.Der DJNZ Befehl ist mit einer relativen Sprunganweisung und einem decrementieren des B-Registers gekoppelt.Dieses hört sich recht kompliziert an ,ist jedoch einfach zu lösen.
Beispiel:

```
06 06       LD   B,06 (hex)
21 00 00    LD   HL,0000 (hex)
23          INC  HL
10 FD       DJNZ FD (springt zu INC HL)
```

Dieses ist eine Schleife von 6 bis 1 ,wobei das HL-Register incrementiert wird und nach Aufführung der Routine den Wert 6 erhält.Beim DJNZ wird das B-Register heruntergezählt.Erhält das B-Register den Wert 0,so wird das Programm fortgesetzt,andernfalls wird mit Hilfe der indirekten Adressierung zu dem durch den DJNZ definierten Wert gesprungen ,und dadurch eine eine Schleife gebildet.

4.8.8) Ein/Ausgabe Befehle

Ein-und Ausgabebefehle senden Werte über die Port,womit man in der Lage ist ,externe Geräte zu bedienen.Es soll hier nicht weiter auf diese Befehle eingegangen werden,da Sie für den MZ-700 Besitzer keine große Praktische Bedeutung besitzen.Erwähnungswert ist vielleicht noch,daß die Adressierung über das C-Register geht,und das es auch Block Aus-eingabebefehle gibt

5) Vorwort zum Anwenderteil

Diese noch recht oberflächliche Behandlung der Z-80 Architektur möge natürlich nicht als Z-80 Buch aufgefasst werden.Es wurden lediglich die wichtigsten Kapitel angesprochen,um das nötige Grungwissen für den nachfolgenden Programmierkurs zu schaffen.Wie schon einmal erwähnt ,ist der Besitz einer ML mit Disassembler eine Grundlage für diesen Abchnitt.
Es werden exemplarische Beispiele für die Programmierung in ML gebracht.
Großer Wert wurde auf die Einführnung wichtiger Unterroutinen gelegt, welche für den Maschinenspracheprogrammierer sich als nützlich erwiesen haben.Sollte jemand noch der Meinung sein,er fühle sich noch nicht sicher genug , an dieses Kapitel anzugehen,so wäre eine weitere Lektüre von Z-80 Bücher wertvoll.Als für mich zweckmäßig hat es sich früher erwiesen, den Versuch zu wagen,fremde Programme zu dokumentieren zu versuchen.Man nimmt sich also ein fremdes Programme ,und versucht mit Hilfe von Büchern und der ML dieses zu verstehen.Versuchen Sie es,der Lerneffekt wird Sie überraschen.

6) Programmierkapitel

6.1.1) Wie printe ich in ML ?

Grundsätzlich wird beim printen eine Monitorroutine benutzt.Ein im Speicher abgelegter Text wird dabei ab der momentanen Cursorposition ausgedruckt. Zuerst muß dieser Text im Speicher stehen.(In Klammern gesetzte Texte beziehen sich auf die ML-SP 3002,also wie ich mit Hilfe dieser ML die Befehle durchführe).Zuerst setzen wir einen beliebigen Text in den Speicher (T from:2000   DEMOPROGRAMM [CR])
Ab der Adresse 2000 sitzt nun den Text DEMOPROGRAMM .Texte müssen immer mit OD (Hex) abgeschlossen sein,damit dieses als Ende des Strings erkannt wird.Der T-Befehl der ML-3002 macht dieses automatisch.Nun haben wir den Text im Speicher und es fehlt nur noch das Programm,welches Ihn ausdruckt. Dazu wird das DE-Register mit der Anfangsadresse des Textes geladen,und wird die Printroutine des Monitors aufgerufen.Wir geben als Programm ein.
(W from:1200   11 00 20 CD 15 00 C3 OE C4 SHIFT&BREAK)
Unserer Programm beginnt nun ab der Adresse 1200 (hex) und hat neun Bytes. Um dieses zu verstehen,müssen wir es im Mnemoniccode lesen können,also disassemblieren.
(D from:1200 to:1210)
Wenn wir das Programm richtig eingegeben haben müßte folgendes auf dem Bildschirm stehen.

```
1200   11 00 20      LD   DE,2000
1203   CD 15 00      CALL 0015
1206   C3 0E C4      JP   C40E
1209   00            NOP
usw.
```

In der ersten Zeile wird das DE-Register mit der Anfangsadresse des auszudruckenden Srings geladen.Anschließend wird die Monitorrotine aufgrufen welche den Text,abgeschlossen mit OD (hex) ,ausdruckt.Das letzte ist ein Sprung zur Maschinensprache,welche mit C40E (hex) Ihren Warmstart hat. Wenn wir das Programm starten wollen,so springen wir jetzt zur Adresse 1200 (hex).(G 1200).Wenn wir alles richtig gemacht haben,müßte der Text ausgeprinted werden,welchen wir ab 2000 (hex) eingegeben haben,und nach Beendigung des Ausprintens müßte sich die ML-3002 wieder melden.

6.1.2) Wie printe ich in ML ?

Im obigen Falle haben wir einen ganzen Text beliebiger Länge ausgedruckt. Aber meistens braucht man nur ein Zeichen auszudrucken.Dafür gibt es eine andere Monitorroutine (Der Leser sollte sich vorher ruhig das Kapitel Monitorroutinen ansehen.),welche ein Zeichen ausprintet.Dieses Zeichen muß im Akku als ASCII Wert stehen.Dann wird die Monitorroutine aufgerufen und das Zeichen wird geprinted,(Wie in Basic ?CHR$(X)).Wir schreiben dazu folgendes Programm.(W from:1200   3E FF CD 12 00 C3 0E C4)
Wenn wir dieses Programm disassemblieren,so müßte folgendes erscheinen.

```
1200  3E FF      LD   A,FF
1202  CD 12 00   CALL 0012
1205  C3 0E C4   JP   C40E
1208  00         NOP
usw.
```

In diesem Falle wird das ASCII-Zeichen FF (hex) ausgeprintet,also ein π. Dieses können wir ja austesten.('G 1200).Ein Tip am Rande.Bevor man ein Programm eingibt sollte man den Speicher freimachen.(K from:1200 to:C000)

6.2) Wie bediene ich die Tastatur

6.2.1) GET in Maschinensprache

Ein ebenfalls sehr wichtiger Bestand eines Programmes ist die Tastaturabfrage.Diese wird mit Hilfe der im Monitor vorhandenen Getabfrage vollzogen.Man ruft hierbei die Getroutin auf,und im Akku steht der Wert des Zeichens,welches man gerade gedrückt hat.Wenn man nichts drückt,so ist der Akkuinhalt null.Um dieses zu zeigen geben wir ein kleines Programm ein, welches die Tastatur abfragt,und das gedrückte Zeichen ausprintet.
(W from 1200   CD 1B 00 28 FB CD 12 00 C3 00 12)
Wir disassemblieren dieses und folgendes Bild müßte sich am Bildschirm
(D from :1200 to: 1210)

```
1200   CD 1B 00       CALL 001B
1203   28 FB          JR   Z,1200
1205   CD 12 00       CALL 0012
1208   C3 00 12       JP   1200
```

Dieses Programm stellt eine Tastaturabfrage dar.In 1200 (hex) steht die Monitorroutine,welche ein Zeichen von der Tastatur in den Akku holt. Dann frage ich ab,ob ein Zeichen gedrückt wurde,wenn nein,dann springe ich wieder zur Adresse 1200.Wenn ein Zeichen gedrückt wurde,so printe ich es aus.Nach dem Ausprinten (Monitorroutine)springe ich wieder an den Beginn des Programmes.Vorsicht,dieses Programm kann man nur mit RESET unterbrechen.

6.2.2) INPUT in Maschinensprache

Eine andere Art ,Zeichen von der Tastatur zu holen ist die Input-Routine, oder im Fachjargon GETLINE.Diese Routine holt eine ganze Ketten von - Asciizeichen,mit dem Zeichen 0D (Hex) abgeschlossen in den Speicher,wo sie weiterverarbeitet werden kann.Zuerst wird das DE Register mit der Adresse geladen,wo später der eingegebene Text stehen soll.
Beispiel:

```
1200 11 00 20      LD   DE,1200
1203 CD 03 00      CALL 0003
1206 C3 0E C4      JP   C40E
```

Mit diesem Programm können Sie einen eingegebenen Text ab der Adresse 2000 ablegen.Wenn Sie das Programm starten (G 1200),so blinkt der Cursor. Sie können nun eingeben was Sie wollen.Wenn Sie Return drücken so finden Sie die eingebene Zeile ab der Adresse 2000 (hex) wieder.(A from:2000)

6.2.3) CURSOR in Maschinensprache

Eine weitere Möglichkeit ,ein Zeichen von der Tastatur zu holen ist die Monitorroutine 09B3 (Hex).Beim Aufruf dieser Routine wird ein Zeichen im Displaycode von der Tastatur geholt.Dabei wartet ein blinkender Cursor solange,bis eine Taste gedrückt ist.Dieses Zeichen steht dann im Display format im Akku.
Beispiel:

```
1200   CD B3 09      CALL 09B3
1203   32 00 D0      LD   (D000),A
1206   C3 0E C4      JP   C40E
```

Eine kleine Routine,welche ein Zeichen einholt,und links oben im Bildschirm hinprinted.

6.3) Dimensionierung

Wie dimensioniere ich in ML ist eine Frage,die jeder ML-Programmierer sich einmal stellen muß,spätestens dann,wenn er eine Anzahl von Daten sinnreich verwalten will.Am meisten werden in ML 16bit Daten benutzt,also wollen wir versuchen,eine solche Tabelle anzulegen und sie zu verwalten.Was ist unsere Intention ?.Wir wollen eine Anzahl von 2Byte Daten parameterabhängig lesen können.Zuerst legen wir uns eine Tabelle mit 4 Daten an. 10F1,1102,1104,1106 (alle Hex).Wir legen diese Daten ab 3000 (Hex) ab. (W from: 3000 F1 10 02 11 04 11 06 11).Würde in Basic DIM X(4) entsprechen.Nun müssen wir eine Routine entwickeln,welche beim Parameter 1 den ersten Datensatz ausgibt,bei 2 den zweiten u.s.w..Der erste Datensatz ist übrigens 10F1 (hex),der vierte 1106 (hex).Unsere Routine soll den Parameter in Akku empfangen und den dimensionierten Wert über HL ausgeben.Beispiel der Akku enthalte 1,dann wird in HL nach Aufruf unserer Routine 10F1 stehen.Die Anfgangsadresse unserer Tabelle ist 3000.Zu diesem müssen wir 2x den Akkuwert addieren,um dann den 16 bit Wert aus dieser Adresse zu nehmen.Nun aber erstmal das Programm.

```
1200   3E 01      LD   A,01      ← Parameter
1202   21 00 30   LD   HL,3000   ← Beginn der Tabelle
1205   3D         DEC  A
1206   87         ADD  A
1207   06 00      LD   B,00
1209   4F         LD   C,A

120A   09         ADD  HL,BC    ← Addition des Parameters zum Tabellenwert
120B   5E         LD   E,(HL)
120C   23         INC  HL
120D   56         LD   D,(HL)
120E   EB         EX   DE,HL
120F   C3 BA 03   CALL 03BA
```

Dieses ist ein kurzes Programm welches dimensionierte Daten nach obiger Aufgabenstellung berechnet.Gehen wir doch einmal schrittweise durch,was dieses Programm macht.Zuerst wird unser Parameter gesetzt,in diesem Falle 1.Dann wird in HL der Beginn der Tabelle definiert.Nun decrementieren wir den Akku.Warum ?.Ganz einfach.Der erste Wert der Tabelle steht in 3000 dieses Wert wollen wir mit den Parameter 01 erhaltem,das heißt,wir dürfen nichts dazuaddieren,der Akku muß also um ein heruntergezählt werden.
Die nächtens Befehle sind nichts weiter als die Bildung eines fiktiven Z-80 Befehles.Wir können das Programm auch noch einaml gedanklich neu schreiben.Es würde wie folgt aussehen:

```
LD   A,Parameter
LD   HL,Tabellenbeginn
DEC  A
ADD  A
ADD  HL,A  <- fiktiv
LD   HL,(HL) <-fiktiv
JP   HL Printout
```

Wir haben nun den DEC Befehl erläutert,und nähern uns nun dem fiktiven ADD HL,A.Dieses ist ein Befehl,welchen wir uns aus mehreren Z-80 Befehlen zusammensetzen müssen.Die vorangegangene Duplizierung des Akkuwertes bewirkt,das zum HL register der Akku zweimal addiert wird.
Sythaktische Formel (LD   HL,(HL + 2*A)   )
Diese Formel ist der ganze Sinn der Dimesionierung und wird dann immer weiter zerlegt,bis nur noch Z-80 befehle vorhanden sind.Wir haben nun den inneren Term der Klammer erläutert und kommen nun zu dem äußeren was wir oben als fiktives LD HL,(HL) definiert hatten,also in der Umgangssprache etwa als 'Lade HL mit dem Inhalt von HL ' sagen würden.Dieses ist auch nur eine Folge von Z-80 Befehlen (siehe oben).Dieses wäre grob umrissen der Zweck der Formel und Ihre Ausführung.Hier noch mal wertvolle Unterroutinen.

```
              ADD HL,A    (addiere Akku zu HL )
C5            PUSH BC
06 00         LD   B,00
4F            LD   C,A
09            ADD  HL,BC
C1            POP   BC
```

```
C9              RET
                LD  HL,(HL)      (Lade HL mit Inhalt von HL)
D5              PUSH  DE
5E              LD    E,(HL)
23              INC   HL
56              LD    D,(HL)
EB              EX    DE,HL
D1              POP   DE
C9              RET
```

Wenn wir nochmal auf unsere erste fitive Betrachtung der Dimensionierung zurückblicken ,so stellen wir
fest,das wir die beiden letzten Unterroutinen benutzen können.Wir haben dann die Möglichkeit diese in einem ML Programm häufig benutzten Routinen immer aufzurufen.Wir schreiben uns also obig beschrieben Unterroutinen an irgendwelche Stellen im Speicher, z.B. (Routine ADD HL,A nach 2000 und LD HL,(HL) nach 2010 (hex).)Wir haben dort dann immer diese Routinen zur Verfügung und können sie auch aus anderen Bereichen anspringen und nutzen. Wir schreiben dann unser Programm wie folgt:

```
1200   3D          DEC  A
1201   87          ADD  A
1202   CD 00 20    CALL 2000
1205   C3 10 20    JP   2010
```

Als Parameter bringen wir also HL (Beginn der Datentabelle) und den Akku (Parameter für die Dimensionierung.Anschließend rufen wir unsere Dimensionierungsroutine in 1200 auf und erhalten den Wert in HL,welcher in der Tabelle an Stelle X steht.(X ist ja im Akku.)
Ein paar Tips für den Anfänger.In unserer Routine in 1200 steht zum Schluß ein JP,wo doch ein CALL und dann ein RET folgen müßte.
Merksatz:Lasse nach einem CALL niemals ein RET direkt folgen,es kostet Platz und man kann dann anstatt CALL einfach JP schreiben.Dieses kommt dadurch,das die Stellung des Stackpointers nicht verändert wird und der Rücksprung einfach ein RET später erfolgt.

6.4) Schleifen

Schleifen sind eine gute Sache,doch hilft einem der DJNZ Befehl bei einer Schleifengröße über 255 nicht weiter.Hier muß man sich eine 16 Bit Schleife erdenken.In diesem Falle brauchen wir ein Register,welches mit läuft.Nun erstmal ein Vorschlag:

```
01 XX XX     LD   BC,XXXX ←Anzahl der Schleifendurchläufe
C5           PUSH BC      ←Schleifenbeginn
XX XX XX     ← Ausführung der Schleife
C1           POP  BC
0B           DEC  BC
78           LD   A,B
B1           OR   C
C2 XX XX     JP   NZ,Schleifenbeginn
```

Wie man sieht,wird hier das BC Register als Zähler benutzt.Zuerst wird jenes mit dem Wert geladen,welcher die Anzahl der Schleifendurchläufe bestimmt.Anschließend wird ein PUSH BC gemacht.Hier wird das BC Register gepushed,also auf den Stack geschobn.Damit wird der Zählwert gerettet, welcher uns auf dem Stack zur Verfügung steht.Anschließend können wir den Ausführungsteil der Schleife einsetzen.als Beispiel sei hier die Routine 003E angedeutet,welche einen Ton erklingen läßt.Lädt man BC mit 0009 ,so wird also neunmal die Routine 003E aufgerufen,es erklingen neunmal ein Ton.(440 MHZ).Anschließend wird durch POP BC der vorher auf den Stack geschobene BC Wert wieder geholt und somit kommt nurnoch die Abfrage ob BC gleich null ist,also das Schleifenende erreicht ist.Dieses erreicht man durch die Kombination der Befehle LD A,B und OR C.Das heißt man lädt den B-Teil von BC in den Akku und odriert ihn mit dem C-Teil. man könnte formell sagen OR C,B.Was bringt uns dieses.Nun erinnern wir uns an das Odrieren.Nur wenn beide Partner 0 haben bekommt das endprodukt der verknüpfung auch 0.Wenn also BC 0000 enthält,so wird 0 mit 0 odriert,was 0 ergibt.Das Zeroflag wird gesetzt und wir können dann im Programm fortfahren.In jedem anderen Falle wird nie das Ergebnis 0 erreicht werden.Das Zeroflag ist nicht gesetzt und man springt dann dorthin ,wo die Schleife Ihren logischen Einsprung hat.

6.5) BINÄRAUSDRUCK DES AKKUS

Hier ein kleines Beispielprogramm,welches den Akkuinhalt als Achtstellige Binärzahl ausdruckt.Dieses ist eine Unterroutine,welche überall im Speicher lauffähig ist.

```
1500   C5         PUSH BC
1501   F5         PUSH AF
1502   06 08      LD   B,08
1504   07         RLCA
1505   F5         PUSH AF
1506   3E 30      LD   A,30
1508   30 01      JR   NC,150B
150A   3C         INC  A
150B   CD 12 00   CALL 0012
150E   F1         POP  AF
150F   10 F3      DJNZ 1504
1510   F1         POP  AF
1511   C1         POP  BC
1512   C9         RET
```

In diesem Falle haben wir die Unterroutine ab der Adresse 1500 stehen. Wenn den Akku also mit FF geladen wird und wird CALL 1500 machen,so wird auf dem Bildschirm 11111111 (Binärcode für FF) ausgedruckt.
(M from 3000: 3E FF CD 00 15 C3 0E C4)
Die Grundannahme des Programmes ist das der Akkuinhalt durch das Carryflag rotiert wird.Das heißt Bit 7 des Akku landet in C-Flag,Bit 6 wird in Bit 7 geschoben,Bit 5 nach Bit 6 usw,das heißt alle Bits befinden sich nachher ein Bit höher und Bit 7 befindet sich immer im C-Flag.Ist das geschobene Bit 1 gewesen,so soll eine 1 ,andernfalls eine 0 gedruckt werden. Dieses muß achtmal gemacht werden,dann haben wir alle Bit durchgeschoben. Nun zum Programm.Zuerst werden durch die PUSH's die Register AF und BC gerettet,da A und B als einzige in der Unterroutine benutzt werden.
Nun laden wir unsere Hauptschleife,welche durch das B-Register bestimmt wird ,mit 8.Anchließend rotieren wir das erstemal den Akku nach links. Alle Bits befinden sich nun eine Position höher.Diesen Akkuwert retten wir nun wieder auf den Stack um ihn beim nächsten Schieben wieder zu Verfügung zu haben.Wir laden nun den Akku mit 30 (hex) das Zeichen für 0 im Asciicode.Wenn nun eine 0 in das Carryflag geschoben wurde springen wir

gleich zum Aussprinten,andernfalls muß ja eine 1 geschobenen worden sein. In diesem Falle zählen wir zur 30 einen dazu und erhalten 31,das Ascii-zeichen für 1 welches dann ausgeprintet wird.Der Rest ist nur noch Routine. Zuerst wird der Akkuinhalt wiedergeholt,welcher vorher auf dem Stack war (durch POP AF).Nun kommen wir zum ende der Hauptschleife,welche in Kombination mit dem B-Register immer durch den DJNZ Befehl gebildet wird. Wir erinnern uns kurz.Wenn das B-Register 0 ist ,wird im Programm fortgefahren,andernfalls wird die Schleife fortgesetzt,also hier achtmal. Nach Beendigung der acht Durchläufe geht das Programm weiter,es werden die Werte von AF und BC durch den POP vom Stack geholt und die Unterroutine ist durchlaufen worden.

6.6) Z-80 ZUSATZBEFEHLE

Nun kommem wir zu einem Thema,welches für Anfänger und besonders den Kennern der Z-80 Maschinensprache neue Perspektiven eröffnet.Es gibt nicht nur 400 Befehle,sondern viel mehr,nämlich über 1000 Befehle.Diese war dem Hersteller Intel wohl selbst nicht bewußt,jedenfalls verschweigt Intel sowie fast alle die sich mit Z-80 befasen dieses Thema. Grundsätzlich betreffen diese Befehle das IX und IY Register,welche nur über 16Bit Adressierung angesprochen werden können.Mit Hilfe der neuen Z-80 Befehle geht es auch mit 8bit Adressierung.Wir erinnern uns: Die Register IX und IY kann man als große Brüder des HL Register auffassen Man erhielt die Befehle durch ein Voransetzen von DD bzw. FD vor den HL-Opcode.Beispiel.23= INC HL :DD 23 = INC IX.Genau so verhält es sich bei den achtbitoperationen von HL,welche also jeweils durch L und H vollführt werden.So ist dann das Lowbyte von IX (Es wird LX genannt) der große Bruder zum L Register.Beispiel. 2D = INC L : FD 2D = INC LY Zu Beachten ist,das die Register LX,LY,HX und HY immer nur Low -und High Byte von IX und IY sind,also Achtbitregister und nicht wie der Name sagen würde ein Sechszehnbitregister.

6.7) BANKSWITCHING

Ein Vorteil des MZ-700 ist das Bankswitiching,also gewisse Speicherbereiche umzutauschen.Dieses geschieht,indem man auf die Ports E0 ff. einfach ein Signal per OUT gibt.Wir wollen dieses an dem Beispiel des Monitors probieren,welche wir vom ROM in RAM lagern wollen,und dann einen Softmonitor haben.Zuerst das Programm.

```
1200    21 00 00    LD   HL,0000
1203    11 00 20    LD   DE,2000
1206    01 00 10    LD   BC,1000
1209    E5          PUSH HL
120A    C5          PUSH BC
120B    D5          PUSH DE
120C    ED B0       LDIR
120E    E1          POP  HL
120F    C1          POP  BC
1210    D1          POP  DE
1211    D3 E0       OUT  (E0),A
1213    ED B0       LDIR
1215    C7          RST  00
```

Hier bedarf es wohl keiner allzugroßen Erläuterung.Wir transportieren den Bereich von 0000-1000 nach 2000 ,schalten dann auf Softmonitor um und transportieren dann den Bereich von 2000-3000 wieder nach 0000 zurück. Zum Schluß springen wir dann in den Softmonitor (RST 00) Selbtverständlich kann man noch mehr Bereiche switchen,doch diesen steht alles im Handbuch.Mit dem Switchen ist der User in der Lage die vollen 64 K zu nutzen.Was das Handbuch allerdings verschweigt ist die Schwierigkeit,den Bereich ab D000 als Speicher zu benutzen.Wie wohl bekannt liegt hier der Bildschirmbereich des MZ-700.Genau dazu liegt der freie Speicher, welchen man ja gerne benutzen würde.Folge:Man muß sich überlegen,wie man dann den Bildschirm und den dahinterliegenden E-Bereich (Tastaturabfrage Casettenroutinen etc.) nutzen wird.Dieses führt zur der Schlußfolgerung, das man einen neuen Monitor braucht,welcher beides gleichzeitig verwaltet,was allerdings sehr kompliziert ist (deshalb hat das S-Basic einen eigenen Monitor).

6.8) NACHWORT

Wir hoffen,das Sie durch das Kapitel ML-Programmierung dem Schritt etwas über Z-80 Maschinensprache zu lernen etwas näher gekommen sind.Natürlich kann dieses Kapitel Sie nicht sofort zu einem Profi der Maschinensprache machen,da die Vielfalt der Möglichkeiten ,welche Maschinensprache bietet hier nur gestreift wurde.Wir hoffen,Ihnen den Einstieg etwas erleichert, und Ihr Interesse für Maschinensprache geweckt,zu haben.

Plottersteuerung von Maschinensprache

Dieses Kapitel soll es Ihnen ermöglichen,den in den MZ-700 eingebauten Vierfarbplotter auch von Maschinensprache aus zu steuern.
Die Bedienung des Plotters vom S-Basic ist ausführlich im Sharp-Handbuch beschrieben.

Auch in Maschinensprache kann man wie im Basic zwischen zwei verschiedenen Plotterbetriebsarten wählen:

1) GRAPHIK MODUS

Im Graphik-Modus kann man auf verschiedenste Weise mit dem Plotter zeichnen.

2) CHARACTER MODUS (TEXT-MODUS)

Im Character-Modus kann man festgelegte Zeichen (Buchstaben,Ziffern,Sonderzeichen) ausdrucken.Dieses Zeichen kann man in verschiedener Größe (26, 40 und 80 Zeichen pro Zeile) ausdrucken.
Der Plotter kann die Zeichen der Character Tabelle (siehe Sharp Handbuch) darstellen.

Falls dem Plotter der Befehl gegeben wird,ein Zeichen,das sich nicht in dieser Tabelle befindet,auszudrucken,wird automatisch in der nächsten Farbe der hexadezimale Code des Characters ausgedruckt.

MONITORROUTINEN ZUR BEDIENUNG DES PLOTTERS

Im Monitor gibt es zwei Plotterroutinen:

a) CALL (01F8H)

Diese Routine gibt den Akkumulatorinhalt auf dem Plotter aus.Der Akku-Inhalt Kann dabei ein Steuerzeichen oder ein Zeichen aus der Character-Tabelle sein. Vor dem Aufruf dieser Subroutine muß also der Akku mit dem gewünschten Inhalt geladen werden.

b) CALL (01A5H)

Diese Routine gibt die einzelnen Bytes einer Stringkette auf dem Plotter aus. Die Zeichenkette kann dabei Steuerzeichen und/oder Zeichen aus der Character-Tabelle enthalten.Die Stringkette muß mit einem ODH beendet sein und der Anfangsspeicheradresse der Stringkette muß im DE-Register gespeichert sein.

Beide Plotterroutinen haben dieselbe Wirkung:Sie geben ein Byte auf dem Plotter aus.Die Routine 01A5H sollte man dann verwenden,wenn man viele Bytes (Zeichen und Steuerzeichen) auf dem Plotter ausgeben will,ansonsten nimmt man die Routine 018FH,die jeweils nur ein Byte auf dem Drucker ausgibt.

STEUERUNG DES PLOTTERS IM CHARACTER-MODUS

Nach dem Einschalten des MZ-700 befindet sich der Plotter automatisch im Character-Modus.Die Druckfarbe des Plotters ist automatisch schwarz und der Druckkopf befindet sich links oben.

Mit Hilfe der beiden Plotterroutinen (CALL (018FH) und CALL (01A5H)) kann man verschiedenen Steuerfunktionen
und Zeichen der Character-Tabelle auf dem Plotter ausgeben.

Steuerzeichen	Bedeutung
01H	Schaltet den Plotter in den Character-Modus
02H	Schaltet den Plotter in den Graphik-Modus
03H	Zeilenrückschub um eine Zeile
04H	Druckertest (4 Quadrate in unterschiedlichen Farben)
0AH	Zeilenvorschub um eine Zeile
0BH	Schriftgrößenwechsel von 40 auf 26 Zeichen pro Zeile

0CH	Schriftgrößenwechsel von 26 auf 40 Zeichen pro Zeile
0DH	Wagenrücklauf
0EH	Druckkopf wird um eine Position nach links bewegt.
0FH	Seitenvorschub
1DH	Wechsel der Farbe
09H,09H,09H	Schriftgrößenwechsel von 40 auf 80 Zeichen pro Zeile
09H,09H,0BH	Schriftgrößenwechsel von 80 auf 40 Zeichen pro Zeile

Weitere Funktionen siehe auch Seite 203 des Sharp-Handbuches.

PROGRAMMBEISPIELE ZUR BEDIENUNG DES PLOTTERS AUS DEM CHARACTER-MODUS

Mit den oben genannten Steuerzeichen,den Zeichen aus der Character-Tabelle und den beiden genannten Monitor-Routinen kann man nun diverse Funktionen ausführen.

Programm 1:

```
LD A,0BH        26 Zeichen pro Zeile
CALL 018FH      Monitorroutine zur Ausgabe des Akku-Inhaltes
LD A,42H        Zeichen 'B'
CALL 018FH
LD A,42H        Zeichen 'B'
CALL 018FH
LD A,47H        Zeichen 'G'
CALL 018FH
```

Dieses kurze Programm schaltet den Plotter auf 26 Zeichen pro Zeile und druckt dann die Buchstaben 'BBG' aus.

Die gleiche Ausgabe kann man auch anders,und zwar mit der zweiten Monitor-Routine,realisieren.

Programm 2:

```
LD DE,A000H     Anfangsadresse der Zeichenkette
CALL 01A5H      Monitorroutine zur Ausgabe einer Zeichenkette
```

Dabei muß in A000H 0BH stehen.
A001H 42H
A002H 42H
A003H 47H

STEUERUNG DES PLOTTERS IM GRAPHIK-MODUS

Folgendes Programm schaltet den Plotter vom Character-Modus in den Graphik-Modus:
Programm 3:

```
LD A,02H        Steuerzeichen für Graphik-Modus
CALL 018FH
```

Nach Ausführung dieses Programmes befindet man sich im Graphik-Modus.
Die Steuerung des Plotters erfolgt ähnlich wie im Character-Modus.Es stehen spezielle Steuerzeichen und Graphikbefehle zur Verfügung.Es werden zur Steuerung ebenfalls die beiden Monitorroutinen 018FH und 01A5H verwendet.
Bei einigen Graphikbefehlen müssen zusätzlich zum Befehl auch noch Parameterwerte an den Plotter übergeben werden.

Allgemeines zu den Parameterwerten:
Alle Parameterwerte sind dezimale Zahlen und können maximal dreistellig sein (Hunderter,Zehner,Einer) .
Wenn ein Befehl mehrere Parameterwerte benötigt,werden die einzelnen Parameterwerte mit einem Komma (ASCII 2CH) getrennt.
Die Parameterwerte können zwischen den Zahlen -999 bis 999 liegen.Bei negativen Werten wird vor dem Parameterwerten ein Minus-Zeichen (ASCII 2DH).Die einzelnen Ziffern der Parameterwerte werden ebenfalls im ASCII-Format angegeben.

Befehle im Graphik-Modus:

a) Befehle ohne Parameterwerte:

Befehl	ASCII	Bedeutung
A	41H	Rückkehr zum Character-Modus
H	48H	Hebt die Mine ab und bewegt den Druckkopf in die Ausgangs-

position.

I 49H Die aktuelle Druckkopfposition wird als Ausgangsposition definiert.

b) Befehle mit einem Parameterwert

Befehl	ASCII	Bedeutung
C	43H	Farbwechsel des Druckkopfes.

Diesem Befehl muß ein Parameterwert zwischen 0 und 3 (ASCII 30H bis 33H) folgen.

0 = schwarz ASCII 30H
1 = blau ASCII 31H
2 = grün ASCII 32H
3 = rot ASCII 33H

Folgendes Programm führt einen Farbwechsel auf rot durch:

```
LD A,43H        C-Befehl
CALL 018FH      Monitorroutine
LD A,33H        Parameterwert Farbe rot
CALL 018FH      Monitorroutine
```

L 4CH Bestimmt Linientyp (durchgehend oder punktiert)

Diesem Befehl muß ein Parameterwert zwischen 0 und 15 (im ASCII-Format) folgen.

0 = durchgehende Linientyp
1-15 = punktierte Linientypen

Folgendes Programm legt den Linientyp 15 fest:

```
LD A,4CH        L-Befehl
CALL 018FH
LD A,31H        Zehnerstelle des Parameterwertes 15 in ASCII
CALL 018FH
LD A,35H        Einerstelle des Parameterwertes 15 in ASCII
CALL 018FH
```

Q 51H Bestimmt die Richtung in der ein Zeichen ausgedruckt werden soll.

Diesem Befehl muß ein Parameterwert zwischen 0 und 3 (ASCII 30H bis 33H) folgen.

0 = aufrecht

1 = rechts liegend
2 = auf dem Kopf stehend
3 = links liegend

S 53H Bestimmt die Zeichengröße
Diesem Befehl muß ein Parameterwert zwischen 0 und 63 (im ASCII-Format) folgen.
Folgendes Programm wechselt die Zeichengröße auf 60:

```
LD A,53H        S-Befehl
CALL 018FH
LD A,36H        Zehnerstelle des Parameterwertes 60
CALL 018FH
LD A,30H        Einerstelle des Parameterwertes 60
CALL 018FH
```

c) Befehle mit zwei Parameterwerten:

D 44H Zeichnet Linien,beginnend mit der aktuellen Druckerkopfposition zu den angegebenen Koordinatenpunkten bis zum Endpunkt (Xn,Yn).
DX1,Y1,X2,Y2,...,Xn,Yn zeichnet eine Linie von der aktuellen Druckerkopfposition zum Punkt (X1,Y1),von diesem Punkt zum Punkt (X2,Y2) bis zum Endpunkt (Xn,Yn).
Der erste Parameterwert (X-Position) muß zwischen -999 und 999 liegen.
Der zweite Parameterwert (Y-Position) muß zwischen -480 und 480 liegen.
Die Parameterwerte müssen durch ein Komma (ASCII 2CH) getrennt werden.
Nach dem letzten Parameterwert muß eine Endkennung (0DH) folgen
Folgendes Programm zeichnet eine Linie von der aktuellen Druckkopfposition zum Punkt (-20,20).

```
LD A,44H        D-Befehl
CALL 018FH
LD A,2DH        -Zeichen in ASCII
CALL 018FH
LD A,32H        Zehnerstelle des X-Parameterwertes in ASCII
```

```
CALL 018FH
LD A,30H       Einerstelle des X-Parameterwertes in ASCII
CALL 018FH
LD A,2CH       Komma (Trennung der beiden Parameterwerte)
CALL 018FH
LD A,32H       Zehnerstelle des Y-Parameterwertes in ASCII
CALL 018FH
LD A,30H       Einerstelle des Y-Parameterwertes in ASCII
CALL 018FH
LD A,0DH       Kennung für Ende der Daten
CALL 018FH
```

J 4AH Zeichnet Linien,beginnend mit der aktuellen Druckerkopfposition zu dem in relativen Koordinaten gegebenen Punkt (X1,Y1) bis zum Punkt (Xn,Yn).
Der Befehl 'J' ist als solches identisch mit dem Befehl 'D'. Allerdings werden bei 'D' absolute Adressen und bei 'J' re- relative Adressen angegeben.
Die X-Parameterwerte müssen ebenfalls zwischen -999 und 999 liegen.
Die Y-Parameterwerte müssen zwischen -480 und 480 liegen.
Die Parameterwerte werden durch ein Komma (2CH) voneinander getrennt und mit einem 0DH abgeschlossen.
Ansonsten siehe D-Befehl.

M 4DH Hebt die Mine ab und bewegt den Druckerkopf zum Punkt (X,Y).
Dem Befehl müssen zwei Parameterwerte,die durch ein Komma (0CH) getrennt werden,folgen.
Dabei dürfen die X-Parameterwerte zwischen -999 und 999,die Y-Parameterwerte zwischen -480 und 480 liegen.
Die beiden Parameterwerte müssen mit einem 0DH beendet werden.
Folgendes Programm bewegt den Druckerkopf an die Position (-1,123):

```
LD A,4DH       M-Befehl
CALL 018FH
LD A,2DH       Minus-Zeichen in ASCII
CALL 018FH
```

```
LD A,31H       Einerstelle des X-Paramerterwertes in ASCII
CALL 018FH
LD A,2CH       Komma (Trennung der Parameterwerte)
CALL 018FH
LD A,31H       Hunderterstelle des Y-Parameterwertes in ASCII
CALL 018FH
LD A,32H       Zehnerstelle des Y-Parameterwertes in ASCII
CALL 018FH
LD A,32H       Einerstelle des Y-Parameterwertes in ASCII
CALL 018FH
LD A,0DH       Kennung für Ende der Daten
CALL 018FH
```

R 52H Hebt die Mine ab und bewegt den Druckerkopf zum Punkt (X,Y). Der Befehl 'R' ist als solches identisch mit dem Befehl 'M'. Allerdings werden bei 'M' absolute Adressen und bei 'R' re- relative Adressen angegeben.
Die X-Parameterwerte müssen ebenfalls zwischen -999 und 999
Die Y-Parameterwerte müssen zwischen -480 und 480 liegen.
Die Parameterwerte werden durch ein Komma (2CH) voneinander getrennt und mit einem 0DH abgeschlossen.
Ansonsten siehe M-Befehl.

d) Befehle mit drei Parameterwerten

X 58H Dieser Befehl zeichnet ein Karthesisches Koordinatensystem. Der erste Parameterwert muß entweder 0 oder 1 sein.Bei 0 wird die Y-Achse,bei 1 wird die X-Achse gezeichnet.
Der zweite Parameterwert ist der Skalenfaktor und muß zwischen -999 und 999 liegen.
Der dritte Parameterwert enthält die Anzahl der Markierungen pro Achse und muß zwischen 1 und 255 liegen.
Dieser Befehl entspricht sonst dem 'AXIS'-Befehl im Basic.
Bei weiteren Fragen siehe Seite 89 im SHARP-Handbuch.
Die einzelnen Parameterwerte müssen durch ein Komma (2CH) getrennt werden und durch ein 0DH beendet werden.

e) Befehle mit verschiedener Anzahl von Parameterwerten

P 50H Dieser Befehl druckt Zeichen aus.Diesem Befehl muß mindestens ein Parameterwert folgen.
Die auszudruckenden Zeichen werden im ASCII-Code als Parameterwerte angegeben.Die einzelnen Parameterwerte müssen nicht getrennt werden,Jedoch muß dem letzten Buchstaben ein ODH folgen.
Folgendes Programm druckt die Buchstaben 'BBG' aus:

```
LD A,50H       P-Befehl
CALL 018FH
LD A,42H       'B'
CALL 018FH     Ausdruck des ersten 'B'
LD A,42H       'B'
CALL 018FH     Ausdruck des zweiten 'B'
LD A,47H       'G'
CALL 018FH     Ausdruck des 'G'
LD A,0DH
CALL 018FH
```

KOMBINATION MEHRERER BEFEHLE IM CHARACTER UND IM GRAPHIK MODUS

a) Character Modus:

Im Character Modus können die einzelnen Befehle ohne eine Trennung miteinander verbunden werden.
Beispielprogramm

```
LD A,01H       Character-Modus
CALL 018FH
LD A,0FH       Seitenvorschub
CALL 018FH
LD A,09H
CALL 018FH
LD A,09H
CALL 018FH
LD A,09H       80 Zeichen
CALL 018FH
LD A,42H       Buchstabe 'B'
```

```
CALL 018FH
LD A,42H        Buchstabe 'B'
CALL 018FH
LD A,47H        Buchstabe 'G'
CALL 018FH
LD A,0DH        Ende
CALL 018FH
```

b) Graphik Modus:

Im Graphik Modus müssen Befehle,die durch ein 0DH beendet werden müssen mit einem Komma (2CH) getrennt werden.Dabei entfällt das 0DH.Lediglich am Ende,also nach dem letzten Befehl,mus ein 0DH angegeben werden.
Beispielprogramm:

```
LD A,02H        Graphik-Modus
CALL 018FH
LD A,49H        I-Befehl
CALL 018FH
LD A,4CH        L-Befehl
CALL 018FH
LD A,30H        Durchgehende Linientyp
CALL 018FH
LD A,43H        C-Befehl
CALL 018FH
LD A,32H        Grüne Farbe
CALL 018FH
LD A,4AH        J-Befehl
CALL 018FH
CALL 018FH
LD A,31H        '1'
CALL 018FH
LD A,30H        '0'
CALL 018FH
LD A,30H        '0'
CALL 018FH
LD A,2CH        Komma
LD A,2DH        Minus-Zeichen
CALL 018FH
```

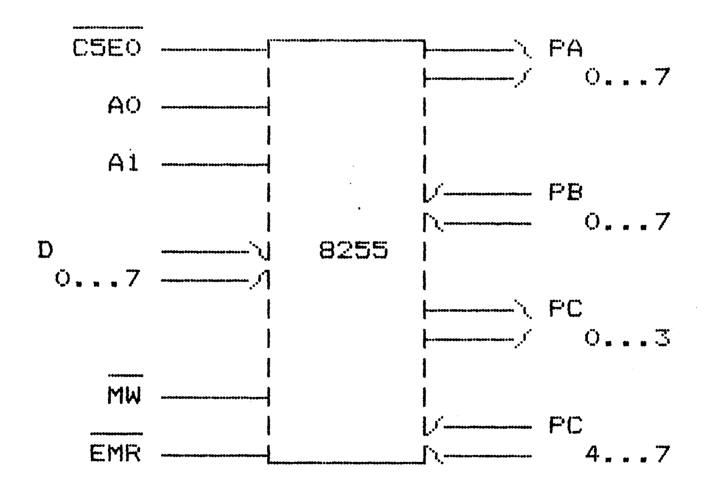
```
LD A,32H        '2'
CALL 018FH
LD A,30H        '0'
CALL 018FH
LD A,31H        '1'
CALL 018FH
LD A,2CH        Komma
CALL 018FH
LD A,30H        '0'
CALL 018FH
LD A,2CH        Komma
CALL 018FH
LD A,30H        '0'
CALL 018FH
LD A,2CH        Komma (Trennung letzter Parameterwert und H-Befehl
CALL 018FH
LD A,48H        H-Befehl
CALL 018FH
LD A,41H        A-Befehl
CALL 018FH
LD A,0DH        Ende
CALL 018F
```

Dieses Programm schaltet den Plotter in den Graphik Modus um.
Dann wird die momentane Druckkopfposition als neuer Ursprung festgelegt.
Der nächste Befehl legt den Linientyp fest,Der nachfolgende Befehl bestimmt die Druckfarbe.
Dann wird eine Linie von den relativen Punkt (100,-201) zum Punkt (0,0) gezogen.
Anschliessend wird der Druckkopf in die Home-Position bewegt und der Plotter wird in den Character-Modus umgeschaltet.
Natürlich kann man diese Programme mit der zweiten Monitorroutine 01A5H wesentlich kürzer fassen.Zur übersichtlichkeit wurden diese Programme jedoch mit der Monitorroutine 01F8H geschrieben.

HARDWARE

Der Portbaustein 8255 hat beim MZ-700 alle Aufgaben zur Steuerung des Cassettenrekorders und zur Abfrage der Tastatur zu übernehmen.

BILD 1: Adressierung und Betriebsart des Portbausteins



Die beim MZ-700 verwendete Betriebsart zeigt das Bild 1.Port A ist auf Ausgabe,Port B auf Eingabe und bei Port C sind die niederwertigen Leitungen auf Ausgabe,die 4 höherwerigen Leitungen auf Eingabe gestellt.
Das geschieht mit dem CONTROL WORD 8AH.Mit diesem Wort muß der MODUS (Betriebsart) eingestellt werden,bevor mit dem Baustein gearbeitet werden kann.Das ROM-MONITOR-Programm macht dies mit der Routine ?MODE bei der Adresse 073EH.

Eine vollständige Darstellung der Möglichen Betriebsarten für den 8255 finden Sie in Tabelle 1.

TABELLE 1: CONTROL WORTE zum 8255 in MODE 0 (Adresse E003H)
A = Ausgabe , E = Eingabe

HEX	CONTROL WORD (Bit) 7 6 5 4 3 2 1 0	PORT A alle Bit	PORT B alle Bit	PORT C Bit 4-7	PORT C Bit 0-3
80	1 0 0 0 0 0 0 0	A	A	A	A
81	1 0 0 0 0 0 0 1	A	A	A	E
82	1 0 0 0 0 0 1 0	A	E	A	A
83	1 0 0 0 0 0 1 1	A	E	A	E
88	1 0 0 0 1 0 0 0	A	A	E	A
89	1 0 0 0 1 0 0 1	A	A	E	E
8A	1 0 0 0 1 0 1 0	A	E	E	A
8B	1 0 0 0 1 0 1 1	A	E	E	E
90	1 0 0 1 0 0 0 0	E	A	A	A
91	1 0 0 1 0 0 0 1	E	A	A	E
92	1 0 0 1 0 0 1 0	E	A	A	E
93	1 0 0 1 0 0 1 1	E	E	A	E
98	1 0 0 1 1 0 0 0	E	A	E	A
99	1 0 0 1 1 0 0 1	E	A	E	E
9A	1 0 0 1 1 0 1 0	E	E	E	A
9B	1 0 0 1 1 0 1 1	E	E	E	E

Nun muß unser MODE-WORT den 8255 aber auch erreichen.Dazu müssen wir uns die Adressen des Bausteins näher ansehen.

Er wird adressiert mit $\overline{\text{CSE0}}$,A0 und A1.Damit ist er ansprechbar mit den Adressen E000H bis E003H.Jede dieser Adressen hat eine bestimmte Bedeutung.

E000 Adresse für PORT A
E001 Adresse für PORT B
E002 Adresse für PORT C
E003 Adresse für CONTROL WORD

Diese Adressen wirken zusammen mit $\overline{\text{WM}}$ für Schreiben und $\overline{\text{EMR}}$ für Lesen. Schreiben bedeutet Ausgabe und Lesen bedeutet Eingabe.

Die Tastaturabfrage

Die PORTS A und B dienen der Tastaturabfrage.Dazu werden über das Ausgabe-Port A die Spalten 1-10 der Tastaturmatrix (siehe Bild 2) nacheinander auf L-Potential gesteuert.Bei jeder Zeilenansteuerung wird bei den Zeilenleitungen 11 bis 18 geprüft,ob das L-Potential über die Martix durchgekommen ist.Aus dem Kreuzungspunkt von Zeilenleitung und Spaltenleitung läßt sich dann die betätigte Taste ermitteln.Diese Art von Tastaturabfrage wird auch 'SCANNEN' genannt.
Natürlich muß nicht immer die gesamte Tastatur gescannt werden;sondern es ist auch möglich nur bestimmte Tasten abzufragen (oder nicht abzufragen) wie z.B. im MONITOR Programm ?BRKEY bei Adresse 0AD2H.
Anhand eines kurzen Programms soll das Zusammenwirken zwischen Hard- und Software erläutert werden.

Zunächst wird an PORT A die Nummer der abzufragenden Spalte (=8) eingestellt mit:

```
        LD A,F8H          3E F8
        LD (KEYPA),A      32 00 E0
```

Zur Stabilisierung wird ein NOP eingefügt.Dann wird über PORT B die Information der Zeilen 11 bis 18 für die Spalte 8 hereingeholt:

```
        NOP               00
        LD A,(KEYPB)      3A 01 E0
```

Rücksetzen des Carry-Bit und Abfrage des Bit 0 (SHIFT):

```
        OR A              B7
        RRA               1F
        JP C,?BRK2        DA 80 09
```

Ist die SHIFT-Taste betätigt (CY=0),wird durch 2x Schieben nach links das BREAK-Bit ins Carry transportiert und BREAK getestet:

```
        RLA               17
        RLA               17
        JR NC,?BRK1       30 04       SHIFT & BREAK betätigt CY=0
```

Ist BREAK nicht betätigt,wird in den ACCU der Wert für die SHIFT-Taste (D6=1) eingeladen,das CY-Flag gesetzt und zurückgesprungen:

```
        LD A,40H          3E 40
        SCF               37
        RET               C9
```

Ist BREAK betätigt wird das CY-Flag gelöscht,das Z-Flag gesetzt und zurückgesprungen.

```
  ?BRK1: XOR A            AF
        RET               C9
```

Dieses Programm teilt also das Ergebnis der Abfrage durch den Zustand der Flags und die Bits 4-6 im ACCU mit.Dabei bedeuten:

a) Z=1 SHIFT & BREAK betätigt
b) D6=1 & CY=1 SHIFT betätigt
c) D5=1 & CY=1 CONTROL (CRTL) betätigt
d) D4=1 & CY=1 SHIFT & CRTL betätigt

Auf die Abfrage der CRTL-Taste im Programmzweig ?BRK2 soll hier nicht weiter eingegangen werden.

In dem vorhergehenden Abschnitt wurde die Abfrage der Sondertasten wie Z.B. SHIFT & BREAK mit einer speziellen Routine behandelt.Das Ergebnis der Abfrage wurde durch die Flags angezeigt.In den meisten Fällen soll aber die gesamte Tastatur abgefragt werden und das Ergebnis der Abfrage als ein CODE-Zeichen in einem Register vorliegen.
Hier hat SHARP einen dem Aufbau der Tastatur angepaßten CODE,den sogenannten DISPLAYCODE,verwendet.
Die eigentliche Tastaturabfrage wird im MONITOR durch das Programm ?SWEP bei Adresse 0A50H durchgeführt.Dieses Programm stellt im Registerpaar BC das Ergebnis der Tastaturabfrage zur Verfügung.Dabei ist das Register B immer 80H und das Register C enthält den Kreuzungspunkt,an dem ein Tastendruck festgestellt wurde.
Der Inhalt von C ist wie folgt zu interpretieren:

XKSS SZZZ

K = SHIFT gedrückt wenn 1
SSS = Nummer der Spalte z.B. Spalte 5 = 100 = 4
ZZZ = Nummer der Zeile z.B. Zeile 11 = 000 = 0

Damit ergibt z.B. die Abfrage der betätigten Taste 'A' im C-Register den Wert 20H oder bei betätigter Taste 'B' den Wert 21H.

Das Tastaturabfrageprogramm des MONOTORS berücksichtigt nicht die Spalte 10=F9H.Dadurch können die Funktionstasten aus dem Monitor nicht abgefragt werden.Das folgende Maschinenspracheprogramm fragt die Funktionstasten F1 bis F5 ab und führt dann einen Sprung zu einer vorprogrammierten Adresse aus:

```
FTASTE: LD A,F9H
        LD (KEYPA),A          KEYPA = E000H
        NOP
        LD A,(KEYPB)          KEYPB = E001H
        CPL
        OR A
        RET Z
        EXX
        LD D,0
        LD B,0
   FT1: INC B
        RLCA
        JR NC,FT1
        LD A,B
        RLCA
        LD E,A
        LD HL,FLEI-2          Ab FLEI liegt Sprungleiste für F1 bis F5.
                              Die Adresse und die einzutragenden Sprung-
                              ziele sind vom Anwender selbst zu wählen.
        ADD HL,DE
        LD C,(HL)
        INC HL
        LD B,(HL)
        POP AF                Stackausgleich
        PUSH BC               Anfangsadresse des Programms
        EXX
        RET                   Start des Programms
```

Bedenken Sie bitte,daß bei diesem Programmvorschlag die F-Tasten-Abfrage eine GET-Funktion hat.Es wird das Programm also nur einmal durchlaufen, d.h. es wird nicht auf den Tastendruck gewartet.Ist keine F-Taste gedrückt kehrt das Unterprogramm mit Z-Flag = 0 zum Hauptprogramm zurück.

Ansteuerung des Kassettenrekorders

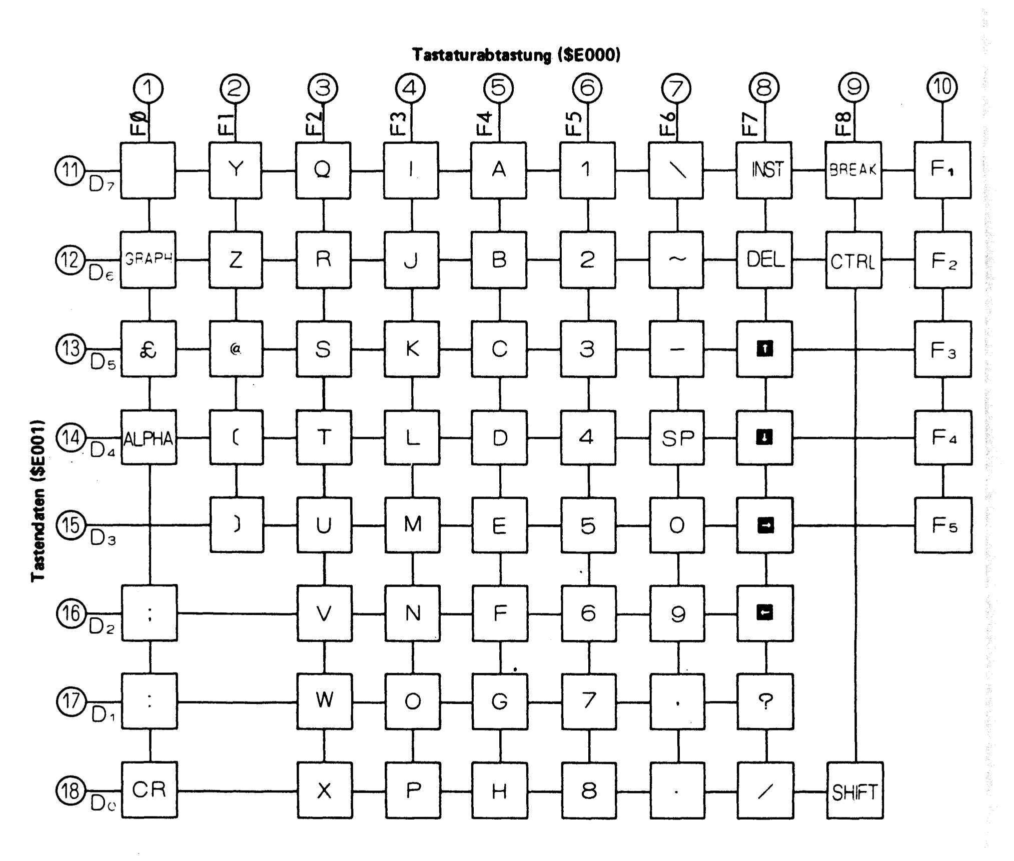
Der Cassettenrekorder wird über das C-PORT des Bausteins 8255 angesteuert. Er hat damit die Adresse E002H.Der C-PORT ist mit dem CONTROL-Wort (8AH) für den 8255 aufgespalten worden.Die Bits 0 bis 3 arbeiten in Ausgabe- und die Bits 4 bis 7 in Eingaberichtung.
Bevor Daten an den Rekorder ausgegeben bzw. von ihm gelesen werden können, muß der Motor gestartet werden.Das kann nur von Hand durch Betätigen der Tasten 'PLAY' oder 'RECORD & PLAY' geschehen.
Der Prozessor fragt lediglich im Monitorprogramm 'MOTOR' bei Adresse 069FH über das Bit PC4 ab,ob der Motor läuft.Dieses Monitorprogramm versucht dann durch einen Schaltimpuls auf Bit PC3 den Motor zu starten.
Dieser Versuch führt nur zum Erfolg,wenn die 'PLAY' oder 'RECORD & PLAY' Taste noch betätigt ist und der Motor vom Programm 'MSTOP' bei Adresse 0700H gestoppt worden ist.Damit ist der Motor per Software stop- und startbar.Das ist besonders dann von Bedeutung,wenn Programmteile bzw. Datenblöcke erst verarbeitet werden müssen,bevor der nächste Block gelesen werden kann.
Die eigentliche Aufzeichnung der Bits erfolgt über die Leitung PC1 zum Rekorder.Eine Programmaufzeichnung erfolgt dabei immer in zwei Teilen;der Information und den Daten.Die Daten sind dabei der eigentliche Inhalt des Programms,die Information (auch mit 'Header' bezeichnet) besteht aus:

1. Filecode	1 Byte zur Kennzeichnug der Art
2. Programmname	16 + 1 Byte
3. Datenlänge	2 Byte
4. Datenanfangsadresse	2 Byte
5. Ausführungsadresse (Startadresse)	2 Byte
6. Kommentar (ohne Bedeutung für die Lade- und Aufzeichnungsprozeduren)	104 Byte

Ein Programm zur Aufzeichnung aus dem Arbeitsspeicher besteht immer aus zwei Teilen:

1. der Aufzeichnung der Information (Header).Diese Aufgabe führt das Monitorprogramm 'WRI' bei Adresse 0021H aus.
2. der Aufzeichnung der Daten.Diese Aufgabe erledigt das Monitorprogramm 'WRD' bei Adresse 0024h.

Bevor das Aufzeichnungsprogramm (SAVE) gestartet wird sollten deshalb im Speicherbereich von 10F0H bis 116FH die für die Information notwendigen bzw. gewünschten Daten abgelegt werden.Das Monitorprogramm 'WRI' zeichnet diesen Datenbereich als Information (Head) auf der Cassette auf.
Beim Lesen der Cassette über Bit PC5 wird dann zunächst die Information (Head) wieder in diesen Speicherbereich zurückgelesen und daraus die Steuerdaten (Anfang,Länge,Startadresse) zum Lesen der Daten entnommen.



Z-80 BEFEHLSÜBERSICHT

ADC A,(HL)	8E
ADC A,(IX+d)	DD 8E 05
ADC A,(IY+d)	FD 8E 05
ADC A,A	8F
ADC A,B	88
ADC A,C	89
ADC A,D	8A
ADC A,E	8B
ADC A,H	8C
ADC A,L	8D
ADC A,n	CE 20
ADC HL,BC	ED 4A
ADC HL,DE	ED 5A
ADC HL,HL	ED 6A
ADC HL,SP	ED 7A
ADD A,(HL)	86
ADD A,(IX+d)	DD 86 05
ADD A,(IY+d)	FD 86 05
ADD A,A	87
ADD A,B	80
ADD A,C	81
ADD A,D	82
ADD A,E	83
ADD A,H	84
ADD A,L	85
ADD A,n	C6 20
ADD HL,BC	09
ADD HL,DE	19
ADD HL,HL	29
ADD HL,SP	39
ADD IX,BC	DD 09
ADD IX,DE	DD 19
ADD IX,IX	DD 29
ADD IX,SP	DD 39
ADD IY,BC	FD 09
ADD IY,DE	FD 19
ADD IY,IY	FD 29
ADD IY,SP	FD 39

AND (HL)	A6
AND (IX+d)	DD A6 05
AND (IY+d)	FD A6 05
AND A	A7
AND B	A0
AND C	A1
AND D	A2
AND E	A3
AND H	A4
AND L	A5
AND n	E6 20
BIT 0,(HL)	CB 46
BIT 0,(IX+d)	DD CB 05 46
BIT 0,(IY+d)	FD CB 05 46
BIT 0,A	CB 47
BIT 0,B	CB 40
BIT 0,C	CB 41
BIT 0,D	CB 42
BIT 0,E	CB 43
BIT 0,H	CB 44
BIT 0,L	CB 45
BIT 1,(HL)	CB 4E
BIT 1,(IX+d)	DD CB 05 4E
BIT 1,(IY+d)	FD CB 05 4E
BIT 1,A	CB 4F
BIT 1,B	CB 48
BIT 1,C	CB 49
BIT 1,D	CB 4A
BIT 1,E	CB 4B
BIT 1,H	CB 4C
BIT 1,L	CB 4D
BIT 2,(HL)	CB 56
BIT 2,(IX+d)	DD CB 05 56
BIT 2,(IY+d)	FD CB 05 56
BIT 2,A	CB 57
BIT 2,B	CB 50
BIT 2,C	CB 51
BIT 2,D	CB 52
BIT 2,E	CB 53
BIT 2,H	CB 54
BIT 2,L	CB 55

BIT 3,(HL)	CB 5E
BIT 3,(IX+d)	DD CB 05 5E
BIT 3,(IY+d)	FD CB 05 5E
BIT 3,A	CB 5F
BIT 3,B	CB 58
BIT 3,C	CB 59
BIT 3,D	CB 5A
BIT 3,E	CB 5B
BIT 3,H	CB 5C
BIT 3,L	CB 5D
BIT 4,(HL)	CB 66
BIT 4,(IX+d)	DD CB 05 66
BIT 4,(IY+d)	FD CB 05 66
BIT 4,A	CB 67
BIT 4,B	CB 60
BIT 4,C	CB 61
BIT 4,D	CB 62
BIT 4,E	CB 63
BIT 4,H	CB 64
BIT 4,L	CB 65
BIT 5,(HL)	CB 6E
BIT 5,(IX+d)	DD CB 05 6E
BIT 5,(IY+d)	FD CB 05 6E
BIT 5,A	CB 6F
BIT 5,B	CB 68
BIT 5,C	CB 69
BIT 5,D	CB 6A
BIT 5,E	CB 6B
BIT 5,H	CB 6C
BIT 5,L	CB 6D
BIT 6,(HL)	CB 76
BIT 6,(IX+d)	DD CB 05 76
BIT 6,(IY+d)	FD CB 05 76
BIT 6,A	CB 77
BIT 6,B	CB 70
BIT 6,C	CB 71
BIT 6,D	CB 72
BIT 6,E	CB 73
BIT 6,H	CB 74
BIT 6,L	CB 75

BIT 7,(HL)	CB 7E
BIT 7,(IX+d)	DD CB 05 7E
BIT 7,(IY+d)	FD CB 05 7E
BIT 7,A	CB 7F
BIT 7,B	CB 78
BIT 7,C	CB 79
BIT 7,D	CB 7A
BIT 7,E	CB 7B
BIT 7,H	CB 7C
BIT 7,L	CB 7D
CALL C,nn	DC 84 05
CALL M,nn	DC 84 05
CALL NC,nn	D4 84 05
CALL NZ,nn	C4 84 05
CALL P,nn	F4 84 05
CALL PE,nn	EC 84 05
CALL PO,nn	E4 84 05
CALL Z,nn	CC 84 05
CALL nn	CD 84 05
CCF	3F
CP (HL)	BE
CP (IX+d)	DD BE 05
CP (IY+d)	FD BE 05
CP A	BF
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP H	BC
CP L	BD
CP n	FE 20
CPD	ED A9
CPDR	ED B9
CPIR	ED B1
CPI	ED A1
CPL	2F

DAA	27
DEC (HL)	35
DEC (IX+d)	DD 35 05
DEC (IY+d)	FD 35 05
DEC A	3D
DEC B	05
DEC BC	0B
DEC C	0D
DEC D	15
DEC DE	1B
DEC E	1D
DEC H	25
DEC HL	2B
DEC IX	DD 2B
DEC IY	FD 2B
DEC L	2D
DEC SP	3B
DI	F3
DJNZ e	10 2E
EI	FB
EX (SP),HL	E3
EX (SP),IX	DD E3
EX (SP),IY	FD E3
EX AF,AF'	08
EX DE,HL	EB
EXX	D9
HALT	76
IM 0	ED 46
IM 1	ED 56
IM 2	ED 5E

IN A,(C)	ED 78
IN B,(C)	ED 40
IN C,(C)	ED 48
IN D,(C)	ED 50
IN E,(C)	ED 58
IN H,(C)	ED 60
IN L,(C)	ED 68
INC (HL)	34
INC (IX+d)	DD 34 05
INC (IY+d)	FD 34 05
INC A	3C
INC B	04
INC BC	03
INC C	0C
INC D	14
INC DE	13
INC E	1C
INC H	24
INC HL	23
INC IX	DD 23
INC IY	FD 23
INC L	2C
INC SP	33
IN A,(port)	DB 20
IND	ED AA
INDR	ED BA
INI	ED A2
INIR	ED B2
JP nn	C3 84 05
JP (HL)	E9
JP (IX)	DD E9
JP (IY)	FD E9
JP C,nn	DA 84 05
JP M,nn	FA 84 05
JP NC,nn	D2 84 05
JP NZ,nn	C2 84 05
JP P,nn	F2 84 05

JP PE,nn	EA 84 05
JP PO,nn	E2 84 05
JP Z,nn	CA 84 05
JR C,e	38 2E
JR NC,e	30 2E
JR NZ,e	20 2E
JR e	18 2E
LD (nn),A	32 84 05
LD (nn),BC	ED 43 84 05
LD (nn),DE	ED 53 84 05
LD (nn),HL	22 84 05
LD (nn),IX	DD 22 84 05
LD (nn),IY	FD 22 84 05
LD (nn),SP	ED 73 84 05
LD (BC),A	02
LD (DE),A	12
LD (HL),A	77
LD (HL),B	70
LD (HL),C	71
LD (HL),D	72
LD (HL),E	73
LD (HL),H	74
LD (HL),L	75
LD (HL),n	36 20
LD (IX+d),A	DD 77 05
LD (IX+d),B	DD 70 05
LD (IX+d),C	DD 71 05
LD (IX+d),D	DD 72 05
LD (IX+d),E	DD 73 05
LD (IX+d),H	DD 74 05
LD (IX+d),L	DD 75 05
LD (IX+d),n	DD 36 05 20
LD (IY+d),A	FD 77 05
LD (IY+d),B	FD 70 05
LD (IY+d),C	FD 71 05
LD (IY+d),D	FD 72 05
LD (IY+d),E	FD 73 05

LD (IY+d),H	FD 74 05
LD (IY+d),L	FD 75 05
LD (IY+d),n	FD 36 05 20
LD A,(BC)	0A
LD A,(DE)	1A
LD A,(HL)	7E
LD A,(IX+d)	DD 7E 05
LD A,(IY+d)	FD 7E 05
LD A,(nn)	3A 84 05
LD A,A	7F
LD A,B	78
LD A,C	79
LD A,D	7A
LD A,E	7B
LD A,H	7C
LD A,I	ED 57
LD A,L	7D
LD A,R	ED 5F
LD A,n	3E 20
LD B,(HL)	46
LD B,(IX+d)	DD 46 05
LD B,(IY+d)	FD 46 05
LD B,A	47
LD B,B	40
LD B,C	41
LD B,D	42
LD B,E	43
LD B,H	44
LD B,L	45
LD B,n	06 20
LD BC,(nn)	ED 4B 84 05
LD BC,nn	01 84 05
LD C,(HL)	4E
LD C,(IX+d)	DD 4E 05
LD C,(IY+d)	FD 4E 05
LD C,A	4F
LD C,B	48
LD C,C	49
LD C,D	4A
LD C,E	4B

LD C,H	4C
LD C,L	4D
LD C,n	0E 20
LD D,(HL)	56
LD D,(IX+d)	DD 56 05
LD D,(IY+d)	FD 56 05
LD D,A	57
LD D,B	50
LD D,C	51
LD D,D	52
LD D,E	53
LD D,H	54
LD D,L	55
LD D,n	16 20
LD DE,(nn)	ED 5B 84 05
LD DE,nn	11 84 05
LD E,(HL)	5E
LD E,(IX+d)	DD 5E 05
LD E,(IY+d)	FD 5E 05
LD E,A	5F
LD E,B	58
LD E,C	59
LD E,D	5A
LD E,E	5B
LD E,H	5C
LD E,L	5D
LD E,n	1E 20
LD H,(HL)	66
LD H,(IX+d)	DD 66 05
LD H,(IY+d)	FD 66 05
LD H,A	67
LD H,B	60
LD H,C	61
LD H,D	62
LD H,E	63
LD H,H	64
LD H,L	65
LD H,n	26 20
LD HL,(nn)	2A 84 05
LD HL,nn	21 84 05

LD I,A	ED 47
LD IX,(nn)	DD 2A 84 05
LD IX,nn	DD 21 84 05
LD IY,(nn)	FD 2A 84 05
LD IY,nn	FD 21 84 05
LD L,(HL)	6E
LD L,(IX+d)	DD 6E 05
LD L,(IY+d)	FD 6E 05
LD L,A	6F
LD L,B	68
LD L,C	69
LD L,D	6A
LD L,E	6B
LD L,H	6C
LD L,L	6D
LD L,n	2E 20
LD R,A	ED 4F
LD SP,(nn)	ED 7B 84 05
LD SP,HL	F9
LD SP,IX	DD F9
LD SP,IY	FD F9
LD SP,nn	31 84 05
LDD	ED A8
LDDR	ED B8
LDI	ED A0
LDIR	ED B0
NEG	ED 44
NOP	00
OR (HL)	B6
OR (IX+d)	DD B6 05
OR (IY+d)	FD B6 05
OR A	B7
OR B	B0
OR C	B1
OR D	B2
OR E	B3
OR H	B4

OR L	B5
OR n	F6 20
OTDR	ED BB
OTOR	ED B3
OUT (C),A	ED 79
OUT (C),B	ED 41
OUT (C),C	ED 49
OUT (C),D	ED 51
OUT (C),E	ED 59
OUT (C),H	ED 61
OUT (C),L	ED 69
OUT (port),A	D3 20
OUTD	ED AB
OUTI	ED A3
POP AF	F1
POP BC	C1
POP DE	D1
POP HL	E1
POP IX	DD E1
POP IY	FD E1
PUSH AF	F5
PUSH BC	C5
PUSH DE	D5
PUSH HL	E5
PUSH IX	DD E5
PUSH IY	FD E5
RES 0,(HL)	CB 86
RES 0,(IX+d)	DD CB 05 86
RES 0,(IY+d)	FD CB 05 86
RES 0,A	CB 87
RES 0,B	CB 80
RES 0,C	CB 81
RES 0,D	CB 82
RES 0,E	CB 83
RES 0,H	CB 84
RES 0,L	CB 85

RES 1,(HL)	CB 8E
RES 1,(IX+d)	DD CB 05 8E
RES 1,(IY+d)	FD CB 05 8E
RES 1,A	CB 8F
RES 1,B	CB 88
RES 1,C	CB 89
RES 1,D	CB 8A
RES 1,E	CB 8B
RES 1,H	CB 8C
RES 1,L	CB 8D
RES 2,(HL)	CB 96
RES 2,(IX+d)	DD CB 05 96
RES 2,(IY+d)	FD CB 05 96
RES 2,A	CB 97
RES 2,B	CB 90
RES 2,C	CB 91
RES 2,D	CB 92
RES 2,E	CB 93
RES 2,H	CB 94
RES 2,L	CB 95
RES 3,(HL)	CB 9E
RES 3,(IX+d)	DD CB 05 9E
RES 3,(IY+d)	FD CB 05 9E
RES 3,A	CB 9F
RES 3,B	CB 98
RES 3,C	CB 99
RES 3,D	CB 9A
RES 3,E	CB 9B
RES 3,H	CB 9C
RES 3,L	CB 9D
RES 4,(HL)	CB A6
RES 4,(IX+d)	DD CB 05 A6
RES 4,(IY+d)	FD CB 05 A6
RES 4,A	CB A7
RES 4,B	CB A0
RES 4,C	CB A1
RES 4,D	CB A2
RES 4,E	CB A3
RES 4,H	CB A4
RES 4,L	CB A5

RES 5,(HL)	CB AE
RES 5,(IX+d)	DD CB 05 AE
RES 5,(IY+d)	FD CB 05 AE
RES 5,A	CB AF
RES 5,B	CB A8
RES 5,C	CB A9
RES 5,D	CB AA
RES 5,E	CB AB
RES 5,H	CB AC
RES 5,L	CB AD
RES 6,(HL)	CB B6
RES 6,(IX+d)	DD CB 05 B6
RES 6,(IY+d)	FD CB 05 B6
RES 6,A	CB B7
RES 6,B	CB B0
RES 6,C	CB B1
RES 6,D	CB B2
RES 6,E	CB B3
RES 6,H	CB B4
RES 6,L	CB B5
RES 7,(HL)	CB BE
RES 7,(IX+d)	DD CB 05 BE
RES 7,(IY+d)	FD CB 05 BE
RES 7,A	CB BF
RES 7,B	CB B8
RES 7,C	CB B9
RES 7,D	CB BA
RES 7,E	CB BB
RES 7,H	CB BC
RES 7,L	CB BD
RET	C9
RET C	D8
RET M	F8
RET NC	D0
RET NZ	C0
RET P	F0
RET PE	E8
RET PO	E0
RET Z	C8

RETI	ED 4D
RETN	ED 45
RL (HL)	CB 16
RL (IX+d)	DD CB 05 16
RL (IY+d)	FD CB 05 16
RL A	CB 17
RL B	CB 10
RL C	CB 11
RL D	CB 12
RL E	CB 13
RL H	CB 14
RL L	CB 15
RLA	17
RLC (HL)	CB 06
RLC (IX+d)	DD CB 05 06
RLC (IY+d)	FD CB 05 06
RLC A	CB 07
RLC B	CB 00
RLC C	CB 01
RLC D	CB 02
RLC E	CB 03
RLC H	CB 04
RLC L	CB 05
RLCA	07
RLD	ED 6F
RR (HL)	CB 1E
RR (IX+d)	DD CB 05 1E
RR (IY+d)	FD CB 05 1E
RR A	CB 1F
RR B	CB 18
RR C	CB 19
RR D	CB 1A
RR E	CB 1B
RR H	CB 1C
RR L	CB 1D
RRA	1F
RRC (HL)	CB 0E
RRC (IX+d)	DD CB 05 0E
RRC (IY+d)	FD CB 05 0E

RRC A	CB 0F
RRC B	CB 08
RRC C	CB 09
RRC D	CB 0A
RRC E	CB 0B
RRC H	CB 0C
RRC L	CB 0D
RRCA	0F
RRD	ED 67
RST 00H	C7
RST 08H	CF
RST 10H	D7
RST 18H	DF
RST 20H	E7
RST 28H	EF
RST 30H	F7
RST 38H	FF
SBC A,n	DE 20
SBC A,(HL)	9E
SBC A,(IX+d)	DD 9E 05
SBC A,(IY+d)	FD 9E 05
SBC A,A	9F
SBC A,B	98
SBC A,C	99
SBC A,D	9A
SBC A,E	9B
SBC A,H	9C
SBC A,L	9D
SBC HL,BC	ED 42
SBC HL,DE	ED 52
SBC HL,HL	ED 62
SBC HL,SP	ED 72
SCF	37
SET 0,(HL)	CB C6
SET 0,(IX+d)	DD CB 05 C6
SET 0,(IY+d)	FD CB 05 C6

SET 0,A	CB C7
SET 0,B	CB C0
SET 0,C	CB C1
SET 0,D	CB C2
SET 0,E	CB C3
SET 0,H	CB C4
SET 0,L	CB C5
SET 1,(HL)	CB CE
SET 1,(IX+d)	DD CB 05 CE
SET 1,(IY+d)	FD CB 05 CE
SET 1,A	CB CF
SET 1,B	CB C8
SET 1,C	CB C9
SET 1,D	CB CA
SET 1,E	CB CB
SET 1,H	CB CC
SET 1,L	CB CD
SET 2,(HL)	CB D6
SET 2,(IX+d)	DD CB 05 D6
SET 2,(IY+d)	FD CB 05 D6
SET 2,A	CB D7
SET 2,B	CB D0
SET 2,C	CB D1
SET 2,D	CB D2
SET 2,E	CB D3
SET 2,H	CB D4
SET 2,L	CB D5
SET 3,(HL)	CB DE
SET 3,(IX+d)	DD CB 05 DE
SET 3,(IY+d)	FD CB 05 DE
SET 3,A	CB DF
SET 3,B	CB D8
SET 3,C	CB D9
SET 3,D	CB DA
SET 3,E	CB DB
SET 3,H	CB DC
SET 3,L	CB DD
SET 4,(HL)	CB E6
SET 4,(IX+d)	DD CB 05 E6
SET 4,(IY+d)	FD CB 05 E6

SET 4,A	CB E7
SET 4,B	CB E0
SET 4,C	CB E1
SET 4,D	CB E2
SET 4,E	CB E3
SET 4,H	CB E4
SET 4,L	CB E5
SET 5,(HL)	CB EE
SET 5,(IX+d)	DD CB 05 EE
SET 5,(IY+d)	FD CB 05 EE
SET 5,A	CB EF
SET 5,B	CB E8
SET 5,C	CB E9
SET 5,D	CB EA
SET 5,E	CB EB
SET 5,H	CB EC
SET 5,L	CB ED
SET 6,(HL)	CB F6
SET 6,(IX+d)	DD CB 05 F6
SET 6,(IY+d)	FD CB 05 F6
SET 6,A	CB F7
SET 6,B	CB F0
SET 6,C	CB F1
SET 6,D	CB F2
SET 6,E	CB F3
SET 6,H	CB F4
SET 6,L	CB F5
SET 7,(HL)	CB FE
SET 7,(IX+d)	DD CB 05 FE
SET 7,(IY+d)	FD CB 05 FE
SET 7,A	CB FF
SET 7,B	CB F8
SET 7,C	CB F9
SET 7,D	CB FA
SET 7,E	CB FB
SET 7,H	CB FC
SET 7,L	CB FD
SLA (HL)	CB 26
SLA (IX+d)	DD CB 05 26

SLA (IY+d)	FD CB 05 26
SLA A	CB 27
SLA B	CB 20
SLA C	CB 21
SLA D	CB 22
SLA E	CB 23
SLA H	CB 24
SLA L	CB 25
SRA (HL)	CB 2E
SRA (IX+d)	DD CB 05 2E
SRA (IY+d)	FD CB 05 2E
SRA A	CB 2F
SRA B	CB 28
SRA C	CB 29
SRA D	CB 2A
SRA E	CB 2B
SRA H	CB 2C
SRA L	CB 2D
SRL (HL)	CB 3E
SRL (IX+d)	DD CB 05 3E
SRL (IY+d)	FD CB 05 3E
SRL A	CB 3F
SRL B	CB 38
SRL C	CB 39
SRL D	CB 3A
SRL E	CB 3B
SRL H	CB 3C
SRL L	CB 3D
SUB (HL)	96
SUB (IX+d)	DD 96 05
SUB (IY+d)	FD 96 05
SUB A	97
SUB B	90
SUB C	91
SUB D	92
SUB E	93
SUB H	94
SUB L	95
SUB n	D6 20

XOR (HL)	AE
XOR (IX+d)	DD AE 05
XOR (IY+d)	FD AE 05
XOR A	AF
XOR B	A8
XOR C	A9
XOR D	AA
XOR E	AB
XOR H	AC
XOR L	AD
XOR n	EE 20

Bedeutung:

nn = Doppeladressbyte oder Doppeldatenbyte
(hier immer als '84 05' dargestellt)

n = Einzeldatenbyte
(hier immer als '20' dargestellt)

d = einzelnes Byte
(hier immer als '05' dargestellt)

ZWEITER ZEICHENSATZ

Hier ist nochmal der neue Zeichensatz abgebildet.Sämtliche Angaben in dezimaler Notation.

SHARP

MZ-800 Serie

Die ausgereifte Personal Computer Serie für Hobby, Ausbildung, Beruf und geschäftliche Anwendungen.

